MORSE: Memory Overwrite Time Guided Soft Writes to Improve ReRAM Energy and Endurance

Devesh Singh
University of Maryland, College Park
United States of America
dsingh19@terpmail.umd.edu

Donald Yeung
University of Maryland, College Park
United States of America
yeung@umd.edu

Abstract

ReRAM is an attractive main memory technology due to its high density and low idle power. However, ReRAM exhibits costly writes, especially in terms of energy and endurance. Prior device studies show that retention can be traded off for write energy and endurance by employing *soft write operations* with lower currents. But given their reduced retention times, soft writes require refresh operations to prevent data loss. Unfortunately, a large number of refreshes are needed in between writes to infrequently updated data. Hence, a non-volatile memory system with soft writes still needs traditional *hard writes*, and a way to choose between them.

The efficacy of soft writes hinges on the time gap between consecutive writes to the same data, which we call the *overwrite time*. If the combined cost of a soft write and its refreshes within the overwrite time window is less than that of a hard write, the original soft write is profitable. Otherwise, it would have been better to perform a hard write to eliminate the refreshes.

To address this, we propose MORSE, a predictor that learns the overwrite times between back-to-back writes to main memory, and associates them with static store instructions in a prediction table. As dynamic stores execute, MORSE predicts the optimal write type based on the predicted overwrite time's magnitude. This soft write decision is placed in the cache hierarchy, and eventually informs the writeback to main memory to use either a soft or hard write. Our results show MORSE provides 2.5x - 4.1x improvement in endurance and 2.9x - 4.2x reduction in write energy over a state-of-the-art predictor. Moreover, we demonstrate that MORSE is within 19.8% of the Oracle policy. Finally, we integrate MORSE with a prior wear leveling technique, called Ouroboros, and show that MORSE improves actual memory system lifetime by 6.2x over a baseline that only employs hard writes.

Keywords

Resistive RAM, Write Energy and Endurance, Soft Writes, Emerging Non-Volatile Memory.

ACM Reference Format:
Devesh Singh and Donald Yeung. 2024. MORSE: Memory Overwrite Time Guided Soft Writes to Improve ReRAM Energy and Endurance. In *International Conference on Parallel Architectures and Compilation Techniques (PACT '24), October 14–16, 2024, Long Beach, CA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3656019.3676890

This work is licensed under a Creative Commons Attribution International 4.0 License.

PACT '24, October 14–16, 2024, Long Beach, CA, USA
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0631-8/24/10
https://doi.org/10.1145/3656019.3676890

1 Introduction

Emerging non-volatile memory technologies, such as resistive random access memory (ReRAM) and phase change memory (PCM), offer much higher memory capacities than DRAM. Additionally, they also improve power efficiency since their non-volatility eliminates the need for refresh operations. Given these benefits, non-volatile memories have been studied quite extensively in the past [2, 34, 35, 44, 58, 72, 76, 87, 92].

The main drawback of non-volatile memories, however, is the high cost of their writes. Not only do their writes incur greater latency than reads, but they also consume significantly more energy and contribute to memory wear out. One approach for addressing this problem is to reduce the frequency of writes to the non-volatile memory. *Hybrid memory systems* take such an approach by employing DRAM alongside the non-volatile memory [1, 11, 17, 20, 21, 31, 58, 64, 65, 69]. For example, the DRAM can be placed in front of the non-volatile memory, acting as a cache that automatically filters the write stream [58].

Rather than reduce the write frequency, another approach for addressing the high cost of writes in non-volatile memories is to sacrifice some retention. Writes to ReRAM or PCM normally employ high currents, long write pulses, and/or multiple write cycles in order to switch the resistive medium (metal oxide for ReRAM or chalcogenide glass for PCM). This ensures the data persistence that is expected for storage systems. (For example, the ReRAM from Crossbar, Inc. has a retention of 10 years [9]). However, in main memory, years-long retention is overkill since the written data will be over-written anyway before the retention time expires.

This opens the door to trade off retention for improved write energy and endurance. In particular, writes to ReRAM or PCM can be performed using lower currents, shorter write pulses, and/or fewer write cycles. Such *soft writes* consume less energy and incur less wear, but they do not fully switch the resistive medium. For instance, a soft write to ReRAM forms a narrower conductive filament (CF) within the metal oxide layer which is susceptible to dissolution [23, 32, 79], and hence, results in lower retention times.

Given their reduced retention times, soft writes require refresh operations to prevent data loss. While the refreshes themselves can also use soft writes, numerous refreshes are needed in between writes to infrequently updated data, which undermines the benefits of using soft writes in the first place. Hence, a non-volatile memory system with soft writes still needs traditional persistent or "hard" writes. Moreover, the memory system also needs a way to decide when to use soft writes and when to use hard writes.

The decision to use soft versus hard writes is governed by a link between write locality and retention time. Specifically, following every soft write, refresh operations are needed periodically in intervals equal to the retention time to avoid data corruption, as shown in Figure 1. Whether soft writes offer any benefits entirely depends on how much time elapses before the same data is written to again–*i.e.*, the time interval between writebacks to the same memory block from the last-level cache (LLC). We refer to this time interval as the *overwrite time*. If the cost of the soft writes occurring within the overwrite time window is less than the cost of a hard write, then the original soft write is profitable. Otherwise, it would have been better to perform a hard write to eliminate the refreshes. For instance, if write energy is the cost objective, then soft writes should be performed whenever the following inequality holds:

$$\frac{OverwriteTime}{RetentionTime} < \frac{E_{HdWr}}{E_{SfWr}} \quad (1)$$

where E_{SfWr} and E_{HdWr} are the soft and hard write energies, respectively. (Inequality 1 is the criterion for selecting soft writes to optimize endurance. Section 2 will discuss optimizing endurance versus total energy in more detail.)

If the overwrite time is known a priori, the optimal soft write decision can be made using Inequality 1. However, for greatest gains, a decision should be made at *every* dynamic store, requiring prediction of the overwrite time for all dynamic store instances. To mitigate the costs of such predictions, *our work makes the key observation that overwrite times are correlated to store PCs*. While overwrite time may vary across different memory accesses, the accesses from the same static store instruction tend to exhibit a single dominant value. Our CPU-side hardware predictor, called *MORSE*, learns the dominant overwrite time per static store PC, and then uses it as a prediction for all dynamic instances of the static store instruction. This permits accurately assessing Inequality 1 for every dynamic store.

One challenge for MORSE is that overwrite times can span multiple soft write retention intervals which can last *several seconds*. A huge number of events occurs across such long time horizons, potentially requiring large predictors to track all of them. However, the stability of overwrite times per store PC mentioned above, coupled with the fact that a few static store instructions typically account for a large fraction of the dynamic stores executed, means that MORSE can employ small buffers. We propose the *Overwrite Time Detector (OTD)* which can learn the most important overwrite times with just 512 entries tracking two dynamic store instances per entry.

To our knowledge, we are the first to exploit retention time tradeoffs for improving write energy and endurance. However, prior work has investigated using soft writes to boost performance. The Region Retention Monitor (RRM) [86] reduces the number of write cycles, and hence the latency, of soft writes compared to hard writes, resulting in performance gains. Unlike our work which uses CPU-side information to guide soft write decisions, RRM employs memory-side information instead. It tracks the number of writes to memory at page granularity, and then employs soft writes for stores to the most frequently written pages and hard writes for all other stores.

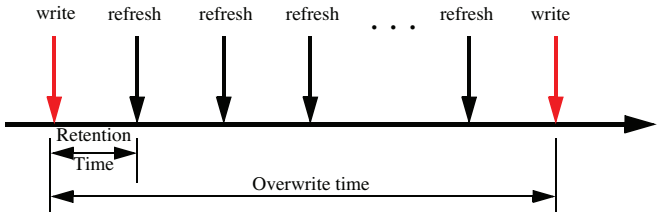

Figure 1: Profitability of soft writes depends on the overwrite time, & hence, the refresh count needed before the next write.

Although RRM was originally used to improve performance, we tried adapting it for write energy and endurance to compare with MORSE. We find that MORSE is much more effective than RRM. RRM's page-level memory-side information is only a proxy for assessing soft write benefits. Inequality 1 is the true determiner of soft write profitability, which MORSE directly assesses for every dynamic store instruction. Moreover, main memory is vast, so RRM's predictor requires more storage and takes much longer to train compared to MORSE which only needs to track a relatively small number of static store instructions. Consequently, RRM misses many soft write opportunities. Our results show MORSE converts 3x more memory writes to soft writes than RRM.

Besides predicting overwrite times, our technique must also issue refreshes to the softly written data. Unlike our prediction mechanism which is performed at the CPU side, tracking the softly written data that must be refreshed can only be done at the memory side. This entails higher overheads since the soft writes can cover a significant portion of main memory. To address this overhead, we integrate refresh with existing wear leveling techniques. In any NVM-based memory system, wear leveling is essential. Commercial wear leveling techniques already keep per-page address translation tables in memory. We propose extending these tables to accommodate MORSE's refresh tracking information. Our work modifies an existing wear leveling technique, Ouroboros [41], but the changes can be extended to other wear leveling techniques too. Although MORSE's refresh information is sizable, it only adds 3.8% more memory on top of Ouroboros' address translation tables.

Our work makes the following contributions:

- We propose a novel hardware predictor, called *MORSE*, to select soft versus hard writes at the CPU side on a per-dynamic store instruction basis.
- We adapt RRM, which was originally designed for performance, to control soft / hard write selection for improving energy and endurance, and compare MORSE against it.
- We propose to integrate tracking of softly written data into existing wear-leveling tables. Also, we adapt wear-leveling to account for writes with varying intensity.
- Finally, we conduct a detailed simulation-based evaluation of MORSE, and compare it against the modified RRM. Our evaluation includes the Oracle algorithm, allowing us to quantify how much of the maximum gain MORSE achieves.

The rest of this paper is organized as follows. First, Section 2 presents the background for our work. Then, Sections 3 and 4 provide the insights behind and the implementation of our technique, respectively. Next, Section 5 discusses experimental methodology, and Section 6 presents our results. Finally, Section 7 discusses related work, and Section 8 concludes the paper.

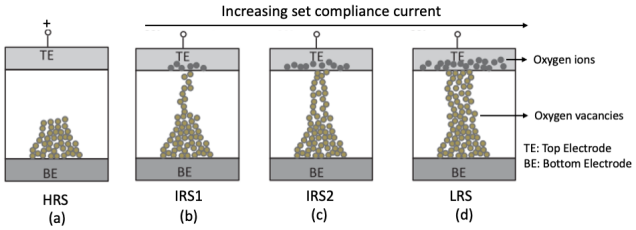

Figure 2: Switching in a ReRAM bitcell (from [73]).

2 Background

A ReRAM bitcell consists of a metal oxide layer sandwiched in between two metal electrodes. In the absence of external influence, this device is in the High Resistance State, or HRS, as shown in Figure 2(a). When a sufficient voltage is applied across the terminals, oxygen ions move from the metal oxide layer to the positive electrode, leaving behind a *conductive filament (CF)* of oxygen vacancies. This *set operation* puts the device in the Low Resistance State, or LRS, as shown in Figure 2(d). The device can switch back to HRS by applying an opposite (negative) voltage to make the oxygen ions travel back into the metal oxide layer and recombine with the vacancies. This *reset operation* dissolves the CF.

After a set operation, the CF tends to naturally dissolve over time (even without an explicit reset) due to migration and drift of oxygen vacancies [79]. The CF's size, or cross-sectional area, determines how long the stored data lasts: the wider the CF, the longer the data lasts, whereas the narrower the CF, the sooner the data is lost [66, 79, 81].

Retention versus Energy / Endurance Relationship. Each set operation forms a CF with cross-sectional area, CF_A, that is *linearly proportional* to the set current, I_{SET}. Likewise, a subsequent reset operation would require a reset current, I_{RESET}, that is also linearly proportional to CF_A in order to dissolve the formed CF [49]. In our work, we assume writes employ a fixed voltage and latency, so the write energy is proportional to I_{SET} and I_{RESET}, and hence, to CF_A.

In addition, the endurance of ReRAM bitcells is inversely proportional to the total write energy absorbed [26, 46, 90]. So, endurance is also inversely proportional to CF_A. In other words, we have the following relationship:

$$CF_A \propto Write\ Energy \propto Endurance^{-1} \quad (2)$$

ReRAM is traditionally deployed as a non-volatile memory, so significant energy is expended during write operations to form stable CFs that last for years. But Expression 2 shows that soft writes can use smaller I_{SET} and I_{RESET}, forming and dissolving narrower CFs, as illustrated by Figures 2(b) and (c), to both reduce energy and increase endurance [49].

As mentioned above, a narrower CF is less stable, and thus exhibits reduced retention time. In particular, the retention time of a ReRAM bitcell is *exponential* with the diameter of the CF, CF_D [32], leading to the following relationship:

$$RetentionTime \propto e^{p*CF_D} \quad (3)$$

where "p" is a device-specific constant. Worst yet, the ReRAM bitcells within a large memory system will be subject to variations, so the actual retention time exhibits a distribution [8, 79]. The retention time for bitcells at the tail of this distribution can be further reduced by up to 1000x [79]. In our work, we use the model from [32] to derive retention time for a given CF size, and then further reduce it by 3 orders of magnitude to account for reliability

Address:	A	B	C	D	.....	A	B	C	D
Page:	P	P	P	P	.....	P	P	P	P
Time:	0	1	2	3	.....	20	21	22	23
Addr OT:	-	-	-	-	.....	20	20	20	20
Page OT:	-	1	1	1	.....	17	1	1	1

OT: Overwrite Time

Figure 3: Overwrite time estimation at line/page granularity.

of the tail bits. This brings the error rate into the range that can be managed by SECDED Error Correction Codes (ECC) [47]. (In this study, we assume ECC stronger than SECDED is impractical for main memory due to the expensive storage and computational overheads).

As an example, consider a soft write that incurs 10x less energy than a basic hard write. As per Expression 2, this will also yield a 10x increase in write endurance. To achieve this, CF_A would need to reduce by 10x as well, leading to a 3.2x reduction in CF_D. Using Expression 2 and factoring in 1000x for the tail bits, there would be a ~52,000x reduction in retention time. We assume a baseline hard write retention time of 10 years, which is based on Crossbar's non-volatile ReRAM technology [9]. So, the retention time for soft writes goes down to 10 seconds. In our work, this 10 second retention time for a 10x energy/endurance improvement is a baseline used in our experiments. However, our evaluation also considers many other tradeoff points (see Section 6.4).

Soft Write Criterion. Inequality 1 provides the criterion for selecting soft versus hard writes by weighing the benefit of soft writes over hard writes against the added refreshes they incur. The former, which we call the *soft write advantage (SWA)*, appears on the right-hand-side of Inequality 1, and changes depending on the optimization aim. Inequality 1 shows the SWA for optimizing endurance–*i.e.*, SWA_{end}. Since endurance is proportional to write energy (Expression 2), SWA_{end} is just the ratio of the hard and soft write energies.

Another optimization objective of interest is total energy. The SWA for total energy, or SWA_{egy}, is not as large as SWA_{end} because the refresh operations shown in Figure 1 must first perform a read (which doesn't affect endurance but does affect energy) before performing a soft write. SWA can also be interpreted as the value at which the overhead of a single hard write is equivalent to a soft write and $(SWA - 1)$ refreshes. For total energy optimization, this gives us:

$$E_{HdWr} = E_{SfWr} + (E_{SfWr} + E_{Read})(SWA_{egy} - 1)$$
$$SWA_{egy} = \frac{E_{HdWr} + E_{Read}}{E_{SfWr} + E_{Read}} \quad (4)$$

where E_{Read} is the read energy. Substituting SWA_{egy} into Inequality 1 yields the criterion for optimizing total energy:

$$\frac{OverwriteTime}{RetentionTime} < \frac{E_{HdWr} + E_{Read}}{E_{SfWr} + E_{Read}} \quad (5)$$

3 Overwrite Time Prediction

Overwrite time is a necessary and sufficient metric for guiding soft writes through Inequality 1 or 5. It is defined per memory location; hence, the spatial granularity at which it is tracked directly impacts its measurement. Tracking it at larger block sizes aggregates writes and artificially reduces the observed overwrite time. For example, Figure 3 shows writes with timestamps to memory/cache blocks A,

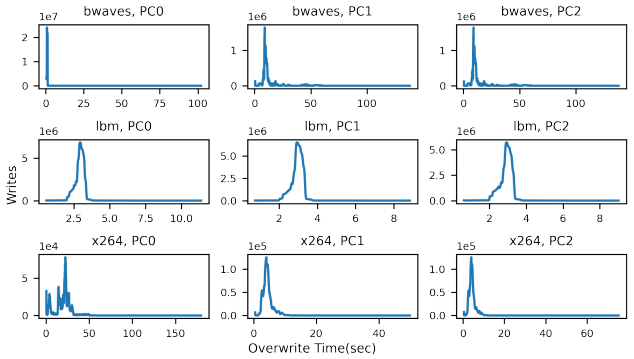

Figure 4: Overwrite time histograms for the top three last-touch stores from three SPEC CPU2017 benchmarks.

B, C, and D that are colocated on page P. The correct overwrite time (at cache block granularity) is 20 for each memory block. However, at page granularity, the observed overwrite times are much smaller: 1,1,1,17,1,1,1. So, to achieve accuracy, overwrite times should be tracked at a fine spatial granularity, however, it is impractical to add overwrite time tracking state for every block in main memory.

To address this problem, we make the key observation that overwrite times are correlated to store PCs. While overwrite times can vary considerably across different dynamic writes to main memory, those linked to the same static store instruction tend to exhibit a single dominant value. To illustrate, Figure 4 shows the overwrite time histograms (at cache block granularity) associated with the top-three static store instructions–PC0, PC1, and PC2–across three different SPEC CPU2017 benchmarks–bwaves, lbm, and x264. Overwrite time correlation around a "peak" is clearly visible in each histogram. While the peak can be wide for some (lbm) or accompanied by secondary peaks in others (x264, PC0), there is nevertheless a single dominant overwrite time in all cases. Hence, there is also a single soft write decision dictated by Inequality 1 or 5 for each static store instruction. (We attribute overwrite times only to the last-touch store, i.e., the store that performs the last write before eviction, assuming writeback caches. All histograms in Figure 4 are for last-touch stores.)

We find such overwrite time correlation stems from memory access patterns on large data structures. For the SPEC2017 benchmarks, there often exists a primary data structure that is updated over and over within a repetitive computation. The data structure is too large to fit in cache, so the scans generate overwrites (via last-touch PCs) to every location in the data structure. Furthermore, each iteration contains similar amounts of work, so the overwrites are periodic, leading to overwrite time correlation. For example, lbm is an iterative solver that refines a double-buffered solution space until convergence; and, x264 performs video compression, encoding batches of video frames into a frame buffer. Both lbm's solution space and x264's frame buffer are overwritten repeatedly as described above, leading to the overwrite time peaks in Figure 4.

Our MORSE predictor exploits this correlation to achieve both high accuracy and low overhead. For each store PC, MORSE observes the overwrite time for a few sampled memory locations. Only a small number of samples are needed due to store correlation, which permits overwrite time measurement at cache block granularity. MORSE uses the sampled overwrite times to assess either Inequality 1 or 5, and then associates the resulting soft write decision back to the static store instruction linked to those writes. When the store instruction executes again, MORSE predicts for every dynamic instance either soft or hard write using the soft write decision previously learned for the corresponding static store. Because there are typically a much smaller number of static store instructions than there are memory locations stored to, MORSE's predictor state is significantly smaller compared to predictors that track memory-side information. (On average, SPEC CPU2017 benchmarks have 4263 unique last touch store PCs, with only 76 accounting for over 90% of writes.)

While MORSE predicts overwrite time at the CPU side from store PCs, refresh (and also reset hard writes discussed later) are tied to specific memory locations, and must be performed at the memory side. Fortunately, the state for these operations does not need to be accessed frequently. MORSE decouples the refresh information from its predictor, moving it into main memory to leverage higher capacities. Later, we show how this information can be integrated with wear leveling to further amortize storage costs.

4 MORSE

This section presents our MORSE technique, that adds several new modules to a multicore CPU. First, the Overwrite Time Detector (OTD) learns the best soft write decision for each static store instruction by observing overwrite times. Second, the soft-write predictor (SWP) uses these learned decisions to predict either soft or hard write for every dynamic store. The SWP is replicated per core, with each SWP predicting for its local core. Figure 5(a) shows how the OTD and SWP are integrated into a multicore CPU. Finally, MORSE maintains information for driving refresh (and reset hard writes) in main memory, which can be integrated with wear leveling.

4.1 OTD

The OTD module, depicted in Figure 5(b), is an associative buffer that trains the SWP modules by observing overwrite times between LLC writebacks. OTD entries are tagged with the PC signature of the store instruction associated with the writeback (label ①, Figure 5(b)). The PC signature consists of core-id bits appended to the PC values. To enable this, we extend a few of the cache tags in the L1, L2, and LLC with a store PC signature field (label ④, Figure 5(d)). Each time a store hit occurs to a tracked cache block in the L1, we create the PC signature for the store instruction and write into this field. Multiple store hits to the same L1 block simply overwrite each other's store PC signature. As dirty L1 cache blocks are evicted, they transfer their store PC signature to the L2 and then to the LLC. Thus, on an LLC writeback, the store PC signature used to access / fill the OTD represents the last-touch store instruction, as discussed in Section 3.

For each PC signature, we store multiple block addresses (or tags) with their timestamps in the OTD. After a PC signature match, its tags are compared against the current writeback address. On a miss, the writeback block address (or tag) is added to the OTD if there is a free entry (label ②, Figure 5(b)). Also, the time when the writeback happened is filled in the OTD along with its tag (label ③, Figure 5(b)). Later on, if the same memory block is written back again, an OTD match occurs, allowing the elapsed time between the two writebacks to be computed by subtracting the previously stored time from the time of the matching lookup. This is the overwrite

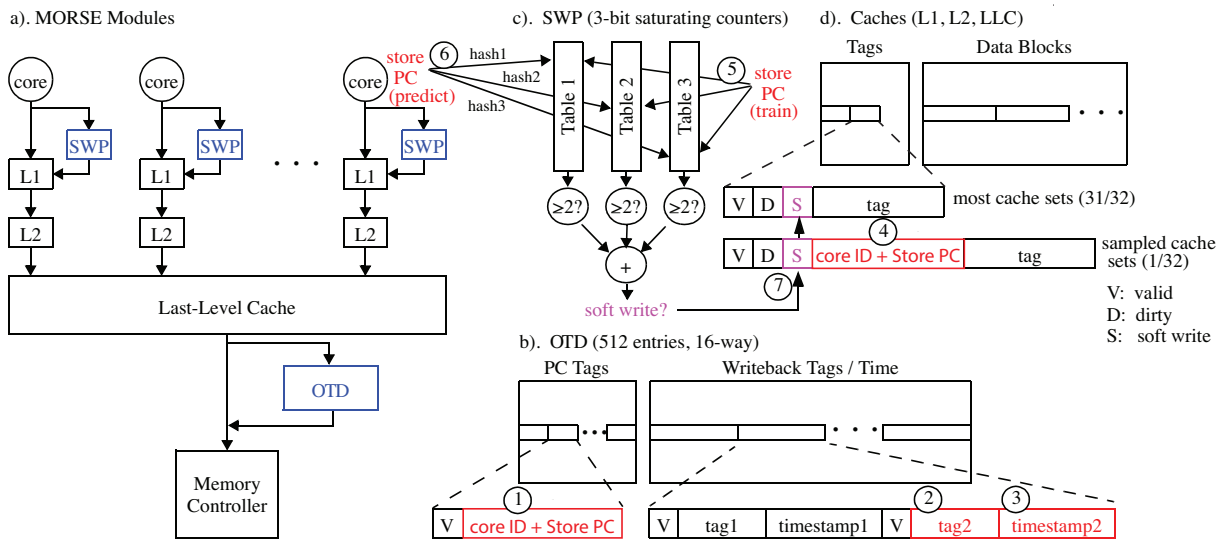

Figure 5: The memory overwrite time predictor (MORSE) (a) adds two hardware modules (in blue) to a multicore CPU: (b) an Overwrite Time Detector (OTD) and (c) a per-core Soft Write Predictor (SWP). (d) MORSE also extends the CPU's caches with soft-write bits and store PC fields.

time which is used to evaluate either Inequality 1 or 5, to determine whether the earlier writeback should have used a soft or hard write. (Note, an OTD lookup involves two matches: the store PC is used to index the OTD and match one of its ways, and then the memory block address is used to match one of the data tags associated with the store PC).

Because the last-touch store PC is associated with each write-back's timestamp, once a match occurs in the OTD and an overwrite time is obtained, we can train the CPU-side (*i.e.*, PC-based) SWP module by associating the resulting soft-write training decision computed from Inequality 1 or 5 with the store PC. The core-id bits from the PC signature are used to identify the target SWP module (label ⑤, Figure 5(c)). The signature can be extended to include bits from process-id to allow targeted training for processes on the same core. In Section 4.2, we will elaborate on this training procedure.

One reason why soft write training decisions are so challenging to learn is because writeback-to-writeback overwrite times can be extremely long, on the order of seconds. In that time, a huge number of writebacks will occur. Fortunately, the OTD only needs to track a tiny fraction of these events for several reasons. First, we find that the majority of dynamic last-touch stores can be traced back to a small number of store PCs. Thus, the OTD only needs to track a few store PCs. For the SPEC2017 benchmarks, we find that 512 store PCs (*i.e.*, OTD entries) are sufficient to capture the "working set" of static store instructions responsible for most last-touch stores.

Second, as discussed in Section 3, most last-touch store PCs exhibit a single dominant overwrite time due to store correlation. Thus, the OTD only needs to sample a small number of overwrite events per store PC. Our OTD tracks just two memory blocks per store PC, as shown in Figure 5(b), and detects only two overwrites at a time. In addition, because the OTD tracks so few memory blocks, we only need a fraction of the cache blocks to carry a store PC field (label ④, Figure 5(d)). To minimize the area overhead in the caches, we keep store PCs in the cache tags for only one out of every 32 sets in the L1, L2, and LLC. We find that both small OTD entries and cache set sampling are sufficient to learn the dominant writeback-to-writeback overwrite time for most store PCs.

Due to the long duration of overwrite events, the memory block tags must remain in the OTD for a long time. Hence, *we never evict memory block tags*. Instead, the tags are removed when a matching lookup occurs, and we complete an overwrite time calculation. We also detect when a tag's timestamp expires–*i.e.*, the timestamp becomes so old that Inequality 1 or 5 (whichever one is being used) is guaranteed to be false even though we have yet to see a reuse occur. Then, instead of waiting for writeback reuse to occur, we conclude that hard write is the correct training decision and eagerly remove the tag. Lastly, memory block tags are also removed when the whole entry is removed due to store PC signature eviction.

4.2 SWP

Each SWP module, illustrated in Figure 5(c), is a predictor that makes a soft or hard write prediction for every dynamic store instruction executed in each CPU core. The predictor we use is adapted from the gskew branch predictor [45]. To mitigate aliasing, three separate tables are employed in each SWP module, each containing a column of 1024, 3-bit saturating counters. Store PCs are used to lookup the predictor. Each lookup indexes into all three tables with a different hash function applied to store PC to compute the per-table indices. (Large tables–1024 entry–ensure collisions may occur in one table but are unlikely to occur in two or more).

As the OTD computes soft write training decisions, it trains the SWP by looking up the 3 counters associated with the tracked store PC (label ⑤, Figure 5(c)) and incrementing or decrementing each one for soft or hard writes, respectively. The OTD output (store PC and soft write training decision) is sent to SWP using the on-chip interconnect between the LLC and L1 caches. The target SWP module is determined using the core-id bits of the PC signature. If the learned training decision is for soft write, the three counters from different tables are incremented; if it is for a hard write, the corresponding counters are decremented. All the counters

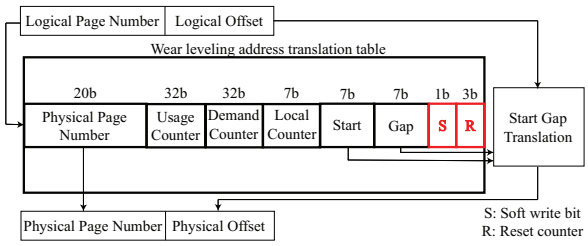

Figure 6: Wear leveling table with MORSE refresh/reset fields(in red).

are initialized to 0, indicating a hard write prediction, but flip to a soft write prediction once the count reaches the selected threshold (2). Thus, two soft write training decisions for the same store PC are needed from the OTD to initiate soft writes for the corresponding static store instruction.

Simultaneously with training for prediction, each CPU core looks up the three SWP tables whenever a store instruction executes using the hashed values of the store's PC (label ⑥, Figure 5(c)). The SWP tables function as three separate voters, each voting for soft or hard writes. If a table's counter value is above the threshold (2), it votes for a soft write; otherwise, it votes for a hard write. The majority vote determines the actual soft/hard write prediction for the store PC being used to index into the tables.

The SWP's soft write prediction must be recorded in the caches so that the correct type of write can be performed upon LLC eviction. We add a *soft-write bit* or "S-bit" to the tags in all of the caches. The SWP writes the S-bit in the L1 tag with its prediction (label ⑦, Figure 5(c)). The S-bit is then copied into the L2 and LLC tags as the cache block is evicted. While we can use set sampling for the store PC signature since it is only involved in training, the S-bit must be carried in all cache tags so that every dirty cache block can be written back with the appropriate write strength (soft or hard).

4.3 Refresh, Resets, and Wear Leveling

In addition to soft write prediction, MORSE must track the memory pages receiving soft writes to refresh the softly written data. Moreover, such tracking is also needed to reset memory locations that cease to receive soft writes. MORSE performs "reset hard writes" to these memory locations to avoid wasteful refreshes. This tracking can be costly due to the large number of softly written pages. Therefore, the tracking data structure is better suited for implementation in main memory. Rather than creating a separate table to track this state, we propose integrating it into wear leveling schemes. Wear leveling is still crucial in soft writable memory systems (soft writes do not affect how the writes are distributed across memory). Yet, to our knowledge, nobody has considered integrating soft writes with wear leveling. In this section, we briefly discuss wear leveling techniques and how we can extend them.

The simplest wear leveling techniques, such as Start-Gap, use algebraic expressions to remap memory locations [22, 39, 57, 60, 68, 78]. This approach is good for small regions, but because the line movements are not targeted, frequently written lines may wear out before relocation happens for larger regions. Alternatively, table-based techniques keep a fully associative mapping from logical to physical blocks along with per-block write counts [14, 30, 61, 83, 92]. These techniques can relocate frequently written regions in a timely fashion, but they incur a larger overhead for the mapping tables. In our work, we consider Ouroboros, a hybrid technique [40]. Ouroboros divides physical memory into regions and uses table-based wear leveling across regions. Within a region, Start-Gap is used. Figure 6 shows Ouroboros's table entry.

Ouroboros uses Start-Gap to shift lines circularly within a region–*i.e.*, a page–after a certain number of local writes (100) tracked by the "local counter." After a fixed number of global writes (50 million), Ouroboros migrates a small number of logical pages (10) with the highest demand to the least used physical pages. The "demand counter" monitors the writes to a logical page since its last migration, and the "usage counter" tracks the overall usage of the physical page.

The additional tracking state we propose has two fields, highlighted in red in Figure 6. This extension to handle refresh and resets only incurs a 3.8% increase in the table size. First, there is a "soft write bit" for each page that indicates whether the page has been softly written. When a page receives a soft LLC write-back, its soft write bit is set. The memory controller checks the soft write bits of a few pages every 10 μsec such that all pages are checked by the end of a retention period. A refresh command is issued for the pages with their soft write bits set. When performing refresh, MORSE refreshes the entire page–it does not further distinguish softly written memory blocks within a page to minimize the tracking overhead.

Resets make the data persistent by reading it and writing it back with hard writes. They are needed when soft writes are no longer beneficial. This might arise due to incorrect soft write predictions or when a program is finished writing to a data structure. Since wear leveling is already tracking frequency of writes to pages, we propose extending it to reset pages that are being refreshed without many writes. We add a 3-bit "reset counter" per page that is incremented when the page is refreshed, and reset to zero when the local counter for start-gap signals a circular shift (100 writes). If the page stops receiving soft writes, the counter continues to increase, indicating that refreshes are being performed without any soft write benefit. When the count reaches a threshold (7), MORSE issues a reset hard write that writes to every memory block in the page, clears the soft write bit, and discontinues refreshing the page.

5 Experimental Methodology

Simulator. Architectural simulators are often used to investigate events that exhibit short time scales, but we are interested in retention events that can last for several seconds. Hence, high-speed simulation is essential for our evaluation. At the same time, cycle accuracy is important too since we need to faithfully model the overwrite times driving soft write decisions. We use Zsim [59] for our evaluation. Zsim is fast enough to simulate the time scales demanded by our research while providing detailed architecture models that enable cycle-accurate simulation. We configured Zsim to model a quad-core CPU with its default beefy out-of-order cores and a 3-level cache hierarchy. The top portion of Table 1 reports the CPU parameters used in our study. Zsim supports the x86 ISA, where CISC instructions are decomposed into simpler micro-ops during the decode stage. MORSE, being a micro-architectural technique, operates on the store micro-ops rather than directly on the

CPU	
Number of cores	4 (x86, out-of-order, 2GHz, 36 entry instruction window, 128 entry reorder buffer, 4 way issue)
L1 Caches	Private, 32 KB I/D, 4-way, 3-cycle hits, 64 B block
L2 Caches	Private, 256 KB, 8-way, 7-cycle hits, 64 B block
L3 Cache	Shared, 16 MB, 16-way, 27-cycle hits, 64 B block
Main Memory	
Memory	667MHz, 8 KB page size, 16 GB capacity
Channels	4
Ranks per channel	4
Banks per rank	8
tRCD (read pulse)	140 cycles (210ns)
tWR (write pulse)	280 cycles (420ns)
tRFC (refresh pulse)	420 cycles (630ns)
Endurance	10^6 hard writes per bitcell
Read Energy	2 pJ/bit
Hard Write Energy	30 pJ/bit
Soft Write Energy	3 pJ/bit
Refresh Energy	5 pJ/bit
Soft write retention	10 seconds
Wear Leveling	Local threshold 10^2 writes, global threshold $5x10^7$ writes, 10 migrations/global epoch, 128 spare frames

Table 1: Simulation parameters.

CISC instructions. Our technique is architecture-agnostic and can be adapted for other ISAs as well.

The main modification we made to Zsim is to add support for MORSE. Specifically, we added the OTD and SWP modules outlined in Section 4. Our experiments model a 512-entry OTD and a 1024-entry SWP, with each SWP module containing 3072 total entries across its 3 tables. Our work also compares against the RRM technique discussed in Section 1. The original RRM paper [86] investigated soft writes for improving performance. We tuned RRM's design parameters to optimize for write energy and endurance. Our Zsim simulator models two versions of this modified RRM (see below).

To facilitate a limit study on soft write techniques, we also implemented an Oracle policy in the simulator. The Oracle records the timestamp for every LLC writeback to precisely compute the overwrite time. When performing a writeback, the simulator retrieves the timestamp for the last writeback to the same memory block and computes the overwrite time. The simulator then uses this overwrite time to determine whether the *previous* writeback should have used a soft or hard write. The simulator then attributes (after the fact) the write energy for either soft or hard write to the previous writeback. In the case of a soft write, the simulator also attributes the write energy for $\lfloor \frac{OverwriteTime}{RetentionTime} \rfloor$ number of refresh operations. Then, the simulator updates the memory block's timestamp with the current time, and continues execution. The Oracle requires a separate region of memory to shadow main memory, providing space for the per-memory block timestamps.

ReRAM Model. Zsim lacks models of non-volatile memory. While there exist NVM simulators, none of them are fast enough for our purposes. Fortunately, Zsim exposes many detailed memory timing parameters that can be configured. We set these parameters to match models available in NVMain [54] to accurately reflect the timing characteristics of a ReRAM-based main memory system. Specifically, we target a base latency (without contention) of 210ns and 420ns for reads and writes, respectively [67]. The middle portion of Table 1 reports the timing parameters we use.

Benchmark	WPKI	Inst (T)	Benchmark	WPKI	Inst (T)
lbm	17.8	7.1	deepsjeng	0.43	7.4
fotonik3d	11.4	6.1	nab	0.36	8
omnetpp	5.5	4.2	parest	0.3	8
roms	3.84	8	namd	0.21	8
mcf	3.49	3.9	perlbench	0.17	2.6
wrf	1.86	8	leela	0.13	8
bwaves	1.62	6.6	blender	0.11	7.1
cam4	1.35	8	x264	0.05	7.7
xz	0.79	2.3	imagick	0.005	8
xalancbmk	0.76	5.1			

Table 2: Benchmarks sorted by Writebacks Per Kilo Instructions (WPKI). Simulated instruction counts are in trillions.

Many different ReRAM energy and endurance numbers have been reported in the literature. Read energies in the range 1-2.4 pJ/bit [50, 67] and write energies in the range 4.8-53 pJ/bit [5, 33, 67, 88] have been reported. We use 2 pJ/bit for reads and 30 pJ/bit for hard writes. Endurance values in between 10^5 and 10^7 write cycles have also been reported [6, 9, 16, 19, 84]. We use an endurance of 10^6 hard writes per bitcell. For soft writes, most of our experiments assume the example presented in Section 2: a 10x improvement in write energy and endurance resulting in a retention time of 10 seconds. The bottom portion of Table 1 reports these baseline energy and endurance parameters we use. However, in Section 6.4, we also explore different energy/endurance versus retention time tradeoff points.

As discussed earlier, MORSE can optimize for either endurance or energy. In our evaluation, we optimize for endurance, using Inequality 1 as the criterion for soft writes. Given the parameters in Table 1, the SWA for endurance (SWA_{end}) is 10x. We have also conducted experiments that optimize for total energy, using Inequality 5. These results are qualitatively similar, though the gains are less since the SWA for energy (SWA_{egy}) is only 6.4x. Due to space limitations, we omit the detailed results for energy optimization.

Lastly, our experiments focus on ReRAM-only memory systems to show fundamental gains. But, MORSE can benefit hybrid memory systems, too. We also investigate their potential impact on MORSE.

Benchmarks. To drive our simulations, we use the SPEC CPU2017 benchmarks [63]. The 19 benchmarks employed in our study represent the complete CPU2017 suite, except for `gcc` and `cactuBSSN` which failed to run on Zsim, and `povray` and `exchange2` which had barely any writebacks. Table 2 lists the benchmarks sorted by memory intensity. We compile them using the Gcc compiler with the highest level of optimization. In every case, we run the benchmark to completion or 8 trillion instructions, whichever occurs first. On average, the benchmarks were simulated for 6.5 *trillion* instructions or a wall clock time of 1,584 seconds. This allows us to observe numerous ReRAM retention time intervals for each benchmark, leading to statistically meaningful results. For our wear leveling experiments, we further extrapolate multiple runs of the benchmarks to determine actual lifetimes. Section 6.3 will describe these experiments.

In our experiments, we study both homogeneous and mixed workloads. For the former, we run the same benchmark on all 4

CPU cores. This allows us to study the effectiveness of our technique given each benchmarks' unique access patterns; yet, it provides a memory access rate appropriate for a multicore memory system. The mixed workloads, covered in Section 6.1, run 4 different benchmarks together, and capture interference effects between different programs.

Hardware Cost. Table 3 shows the parameters and overhead for MORSE. Each SWP table is the same as the local predictor in a tournament branch predictor [29]. We use McPat [37] to get the area and power of this predictor at the 22nm node. By extending these numbers for three tables we get 0.028mm^2 area, 2.6mW dynamic power and 15mW leakage for each SWP module. Similar calculations are undertaken for OTD using Cacti [38]. The OTD module consumes 0.43mm^2 area and 20mW of leakage power. It incurs almost no dynamic power, averaging 60μW for the SPEC benchmarks, as sampling ($\frac{1}{32}$ of the LLC writebacks) eliminates most of the activity.

Table 4 shows the parameters and area for RRM versions. "RRM-base" has 4,096 entries, requiring a 98 KB table, matching the hardware from the original work [86]. "RRM-aggr" is a more aggressive version of the scheme with 8x more entries–32,768–and 8x the table size–784 KB. Both RRM-base and RRM-aggr consume more SRAM than MORSE, which only requires 73.2 KB as shown in Table 3. MORSE also requires 512 KB to track refreshes and resets. However, unlike RRM's table which is used for both prediction and refresh, MORSE's 512 KB table is decoupled from prediction, and is thus accessed infrequently. So, it can be implemented off-chip in non-volatile memory. Moreover, if we integrate MORSE with Ouroboros, the refresh/reset bits can be folded into the wear leveling tables with minimal overhead (Section 4.3). The parameters for both schemes were tuned using comprehensive sensitivity studies. However, due to space constraints, we have omitted results for sensitivity to sampling, MORSE parameters, etc.

In addition to the SRAM overhead, the writeback request packets from the sampled cache blocks must carry the PC signature. This would necessitate extra wires to transmit the PC signature alongside the data and address being written back, adding 6 bytes to a 70-byte wide channel. While we could reuse the existing wires used for transferring PC information for prefetching up to the L2 cache, additional wires would still be required beyond the L2 up to the OTD. However, since the OTD is located immediately after the LLC, the primary additional wiring overhead is between the L2 and LLC. Given that this only applies to writebacks from sampled cache blocks, we opted for a design that introduces an additional cycle for these writebacks from L2 and LLC to transfer the PC signature, rather than adding more wires. The performance impact of the additional writeback cycles and the extra writes introduced by MORSE is discussed in Section 6.2.

OS impact. Since Zsim lacks full-system simulation capabilities, a shortcoming of our work is the inability to simulate context switches. Events like exceptions and interrupts are not a major issue as they are infrequent. However, context switches due to time-sharing on each core could slow down individual programs, impacting overwrite times. Usually, scheduling policies like a completely fair scheduler (CFS) in Linux allocate equal time slices to processes weighted by their priority. This will increase the overwrite time for a process by a predictable amount, allowing our scheme to learn it. But, we leave a detailed study of the impact of time sharing as future work.

OTD		SWP	
Sets	32	SWP count	4
PC Associativity(20 bits)	16	Saturating counter entry size	3 bits
Addresses/PC(48 bits)	2	#Saturating counter entries	1024
Timestamps/PC(32 bits)	2		
SRAM Overhead	**11.8KB**	SRAM Overhead	**4.5KB**
Caches		Wear Leveling Table	
Soft Write Bit (S)	1/entry	Soft write bit	1/entry
PC signature (22 bits)	1/32 entries	Reset counter (3 bits)	1/entry
SRAM Overhead	**57 KB**	NVM Overhead	**512KB**
Total Overhead	**73.2 KB** SRAM and **512 KB** NVM		

Table 3: MORSE parameters and overheads.

RRM-base		RRM-aggr	
Sets	256	Sets	2048
Associativity	16	Associativity	16
Hot threshold	4	Hot threshold	4
Decay interrupt	100/16 sec	Decay interrupt	100/16 sec
Overhead	**98 KB**	Overhead	**784 KB**

Table 4: RRM parameters and overheads.

Another impact of context switching is increased interference in common structures like the OTD. To address this, future work could include process ID bits in the PC signature (as mentioned in Section 4.1), enabling stores from different processes on the same core to co-exist in the OTD. Additionally, increasing the OTD size is feasible since it accounts for only 16% of MORSE's SRAM overhead. We do simulate multiple benchmarks running together on their own cores to study the impact of interference on common structures like OTD in Section 6.1.

6 Experimental Evaluation

This section presents our results, starting with MORSE's improvement in endurance, which is our target objective function. Figure 7 presents the endurance results across our benchmarks. It plots the total write energy for the soft write technique normalized to the total write energy when all writes are hard writes. We call this the *effective* SWA_{end}. This metric is similar to SWA_{end} discussed in Section 2, but applied to all the writes of a benchmark instead of an individual write. It quantifies the wear out improvement provided by the soft write technique. If wear-leveling performance remains unchanged, effective SWA_{end} translates to memory lifetime improvement by the same factor. (Given the parameters from Table 1, the maximum effective SWA_{end} is 10x, when every write is a soft write with no refreshes). As Figure 7 shows, MORSE achieves an effective SWA_{end} of 6.1x on average. This is within 19.8% of the Oracle's effective SWA_{end} of 7.6x. In comparison, RRM-base and RRM-aggr achieve effective SWA_{end} of only 1.5x and 2.4x, respectively.

To provide more insights, Figure 8 breaks down the different types of writes performed by the soft write techniques. Alongside

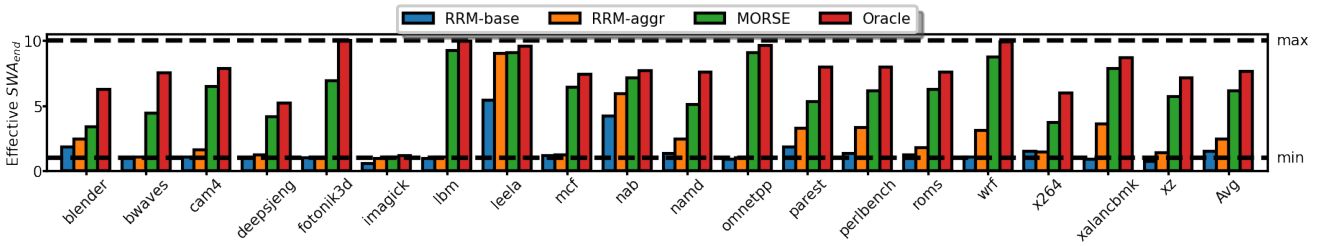

Figure 7: Effective SWA_{end} for different soft write techniques and the Oracle.

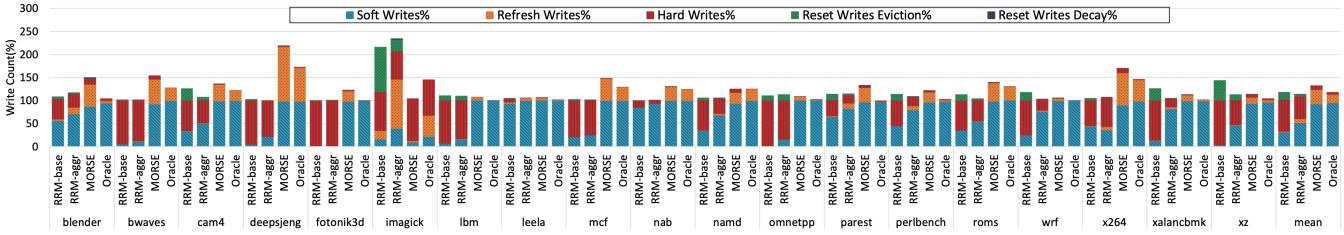

Figure 8: Breakdown of different types of writes normalized against the writes without soft write techniques.

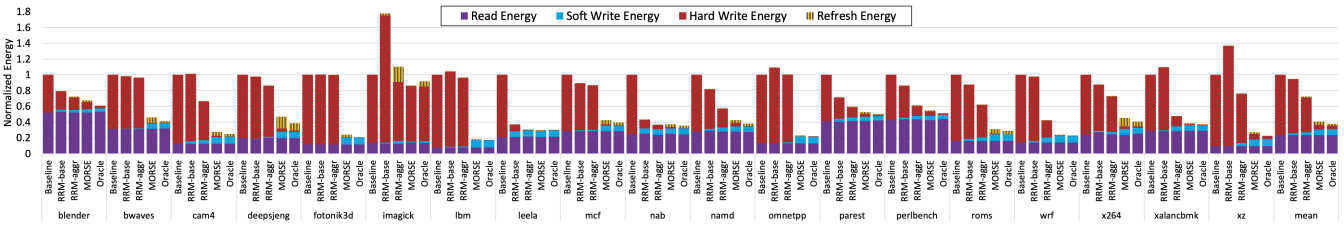

Figure 9: Breakdown of memory system energy normalized against main memory energy with only hard writes.

the soft write, refresh, and hard write categories, it categorizes reset hard writes into "decay" and "eviction". The former counts resets that occur when a technique deems it unprofitable to continue issuing soft writes. The latter accounts for resets due to a shortage of tracking entries. All bars are normalized to the total writebacks by the application. (Since refreshes and resets are in addition to the applications' normal writebacks, the breakdowns add up to more than 100%).

As Figure 8 shows, MORSE does two things very well. First, it identifies and converts a large fraction of writes into soft writes. On average, 91.6% of the application-level writes for MORSE are soft writes, compared to 30% and 51% for RRM-base and RRM-aggr, respectively. MORSE nearly matches the Oracle, where 95% of the application-level writes are soft writes. Second, MORSE exhibits minimal reset hard writes, just 0.4%, while the RRM policies have many more reset hard writes. Specifically, RRM-base and RRM-aggr incur 16% and 4.8% resets, respectively, primarily due to evictions. The Oracle, being omniscient, has zero resets by definition.

MORSE uncovers practically all soft write opportunities by accurately predicting overwrite time, and applying the soft write criterion–Inequality 1–to precisely determine the profitability of soft writes in a fine-grain manner per writeback. This mimics the Oracle's behavior, allowing soft write decisions by MORSE to exhibit high average accuracy, precision, and recall [15] of 95.6%, 99.6%, and 92%, respectively. In contrast, RRM's soft write decisions based on coarse-grained page access counts are inaccurate, missing many soft write opportunities. For instance, we find that many soft write candidates may not land on a hot page, particularly when access patterns are sparse. These candidates may still individually warrant employing soft writes.

Moreover, MORSE achieves its higher accuracy with lower overhead. The OTD consumes only 11.8 KB, which is 8 and 67 times smaller than the hardware tables used in RRM-base and RRM-aggr, respectively. Despite its small size, the OTD can track almost all last-touch store PCs without incurring many evictions. In contrast, RRM (especially RRM-base) exhibits numerous evictions, as seen from the reset eviction components in Figure 8. Overall, MORSE's CPU-side predictor is much lighter weight compared to RRM's memory-side predictor, given the greater number of memory locations written to compared to the store PCs writing them.

Lastly, MORSE's lighter weight mechanism also makes it more adaptive. Even if RRM were able to track every page in memory, it would still need to *train on every page*. Inevitably, some hard writes would sneak through due to long training times. In contrast, MORSE only needs to train on the last-touch store PCs, which can happen rapidly because there are so few of them. This allows MORSE to make correct soft write decisions much sooner.

Despite explictly optimizing for endurance (using Inequality 1 instead of Inequality 5), MORSE still saves energy. Figure 9 presents our energy results by plotting the total energy incurred in the memory system normalized to the energy consumed by the hard writes only baseline system. The figure also breaks down the total energy into reads and various types of writes. (Reset hard writes are folded into the hard write category). Even though there are 4.5x

Mixed workload	Benchmarks
mix1 (4 High)	lbm, fotonik, omnetpp, roms
mix2 (4 Medium)	bwaves, cam4, xz, xalancbmk
mix3 (4 Low)	imagick, blender, x264, leela
mix4 (2 High + 2 Low)	lbm, fotonik, imagick, blender
mix5 (2 High + 2 Low)	omnetpp, roms, x264, leela
mix6 (2 High + 2 Medium)	lbm, fotonik, bwaves, cam4
mix7 (2 High + 2 Medium)	omnetpp, roms, xz, xalancbmk
mix8 (2 Medium + 2 Low)	bwaves, cam4, imagick, blender
mix9 (2 Medium + 2 Low)	xz, xalancbmk, x264, leela

Table 5: Mixed multiprogrammed workloads.

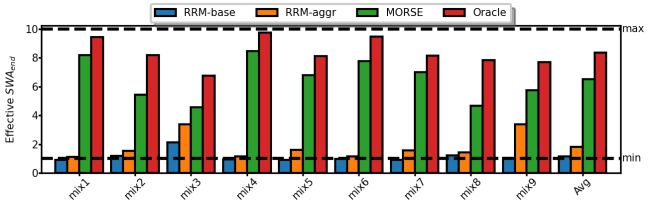

Figure 10: Effective SWA_{end} for mixed workloads.

fewer writes than reads, writes still consume 76% of the memory system energy, on average, as the breakdown of the "Baseline" bars in Figure 9 shows. For the most write intensive benchmark, lbm, writes account for 92% of the energy. However, on average, RRM-base provides almost no reduction in energy because it incurs many reset hard writes due to evictions that nullify its soft write gains, as discussed above. RRM-aggr does better, providing a 27% energy reduction on average since its larger hardware table identifies more soft writes. In contrast, MORSE reduces memory system energy by 2.4x thanks to a 4.5x reduction in write energy. This puts MORSE within 8% of the Oracle.

6.1 Mixed Multiprogrammed Workloads

Thus far, we have only considered individual benchmarks (the same benchmark runs on all 4 cores). We now show experiments using mixed multiprogrammed workloads. To create the workloads, we grouped the benchmarks in Table 2 into high, medium, and low write intensity (WPKI) categories, and then picked 4 different benchmarks at a time with varying write intensities. Table 5 shows the nine multiprogrammed workloads that we created and the mix of write intensities they represent. Figure 10 plots the endurance results for these workloads in the exact same format as Figure 7.

Figure 10 shows, on average, the effective SWA_{end} for both MORSE and Oracle increases from 6.1x and 7.6x to 6.5x and 8.4x, respectively. MORSE is within 22.6% of the Oracle, down from 19.2% in Figure 7, primarily due to interference from multiple workloads. Also, MORSE is 5.8x and 3.6x better than RRM-base and RRM-aggr, respectively, up from 4.1x and 2.5x in Figure 7. MORSE's performance increases with write intensity of the benchmarks while the opposite it true for RRM. In a mix, write intensive benchmarks dominate as they contribute the majority of writes. Hence, MORSE performs even better for these workloads compared to RRM. Although we omit the detailed results to save space, energy is also better for the mixed workloads, with MORSE reducing total main memory energy by 3x.

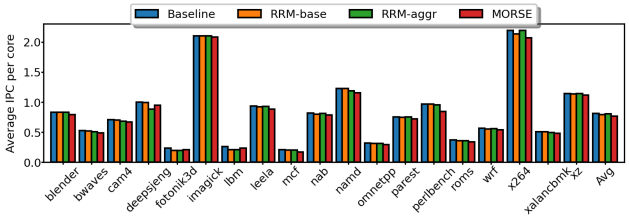

Figure 11: Performance impact of soft write techniques.

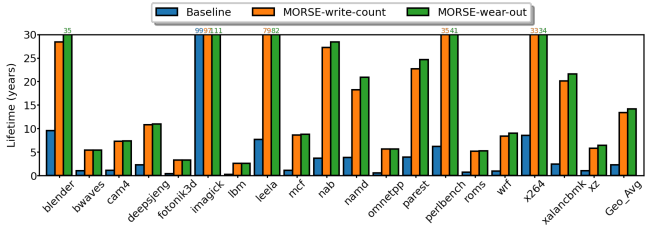

Figure 12: Extrapolated lifetime

6.2 Performance Results

Soft write techniques improve energy and endurance, but they also add refreshes and resets. While these additional memory operations are off the CPU access's critical path, they can still affect performance by increasing contention in the memory system. Figure 11 reports the performance (average IPC per core) for the techniques we evaluate. Remarkably, all soft write techniques incur a very small performance penalty, indicating minimal contention effects. MORSE's performance hit is the largest, at 5.3%, due to its aggressive approach in issuing soft writes, which leads to the highest number of refreshes. This impact is further compounded by the additional writeback cycles from the sampled L2 and LLC cache blocks, as discussed in Section 5. But, MORSE's performance impact is minimal relative to the energy and endurance gains it provides.

6.3 Wear Leveling Results

Next, we evaluate MORSE integrated with Ouroboros wear leveling, and use the combined system to quantify actual lifetime. Since Start-Gap has been shown to be highly effective in regions much larger than our pages, we assume perfect wear leveling within a page, and extrapolate to examine wear leveling across pages by Ouroboros. For extrapolation, we collect per-page write profiles for each benchmark at Ouroboros' global migration epochs (every 50 million writes). We wrote a separate program to perform the extrapolation which repeatedly applies the benchmark's write profiles and runs Ouroboros' page migration algorithm until all spare frames (128) are worn out. The number of "runs" of the benchmark during extrapolation and the runtime for a single simulated run yields the memory system lifetime if the program was run repeatedly until failure.

When integrating with wear leveling, we propose to use wear out as a guiding metric instead of write count given that soft and hard writes inflict uneven wear. Figure 8 shows that for MORSE, hard writes account for only 6.3% of the writes, but they are responsible for 23% of the wear. If we don't distinguish soft versus hard writes, the wear leveling technique will perform the wrong page migrations. To make Ouroboros consider wear out, its usage

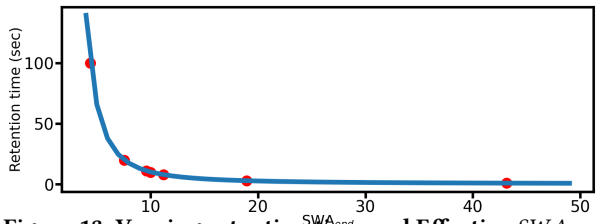

Figure 13: Varying retention time and Effective SWA_{end}.

counters are incremented by 1 for soft writes and by SWA_{end} (10) for hard writes.

Figure 12 compares MORSE's lifetime against the baseline with just hard writes. MORSE's lifetime accounts for wear leveling via extrapolation and the baseline assumes perfect wear leveling. (So, the gains of MORSE over baseline are conservative). For MORSE, we report the lifetime when using the write count metric versus the wear out metric to guide wear leveling, labeled 'MORSE-write-count' and 'MORSE-wear-out,' respectively. On average, we see 6.8% increase in lifetime–from 13.3 years to 14.2 years–when using the wear out metric. Overall, MORSE improves average lifetime of ReRAM from 2.3 years (for the baseline) to 14.2 years, a gain of 6.2x. Overall, the 6.1x effective SWA_{end} translates to 6.2x improvement in memory lifetime. The difference stems from the slight performance degradation.

6.4 Optimal Retention Time

Thus far, we have only considered a single energy versus retention time tradeoff point (the one in Table 1). However, many tradeoff points exist, with the best one dependent upon the overwrite time values of a program. For instance, a program with very short overwrite times would benefit from a more aggressive reduction in CF_A, resulting in reduced retention time but significantly higher SWA_{end}.

Figure 13 plots various retention times versus SWA_{end}. The lower limit of the sweep is constrained by the ReRAM cell's operational integrity. Beyond a certain point, the filament would cease to be continuous, which is assumed implicitly in the device-level models, making the ReRAM cell non-functional. Experimental data is available supporting ReRAM operation at 10x reduction in CF_A, which is the baseline for this work [79]. It is difficult to estimate the extent to which CF_A can be further decreased before the ReRAM cell ceases to function. We assume 1 second as the lower limit for retention time where CF_A is reduced by 43.13x, based on the observation that the model's results plateau around that value in Figure 13.

We swept the retention time from 1 to 100 seconds, and used the Oracle policy to identify the best one per benchmark. (By recording every overwrite time, the Oracle can evaluate the SWA benefit against the refreshes incurred for a given retention time, allowing identification of the optimal retention time). We found most benchmarks prefer 1 second except for fotonik, bwaves, deepsjeng, x264, and imagick whose optimal retention times are 3, 11, 20, 8, and 100 seconds, respectively. (The upper limit was set to 100 since all benchmarks, except imagick, plateau before reaching that point).

Figure 14, shows the results when using the best retention time for each program, denoted as 'MORSE-opt' and 'Oracle-opt'. On average, the Oracle improves to 15.39, a gain of 201% compared

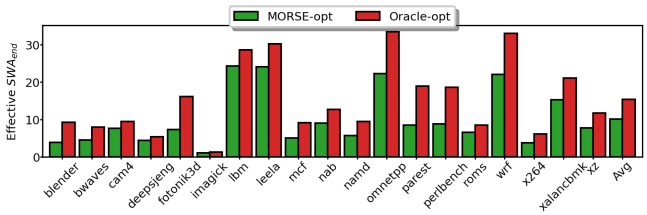

Figure 14: Effective SWA_{end} for MORSE and Oracle with optimal retention per benchmark.

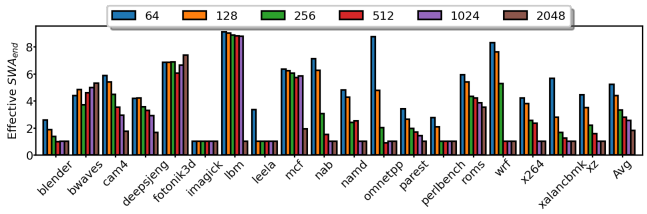

Figure 15: Effective SWA_{end} for MORSE vs the LLC size(MB).

to the earlier evaluated baseline. MORSE-opt improves to 10.48, a gain of 166% over its baseline. The primary reason for MORSE-opt not achieving as much improvement as Oracle-opt is because of higher importance of refresh at lower retention times. Although the accuracy of the scheme remains consistent across different retention times (91.5% average), refreshing at a coarse page granularity starts to have a more detrimental effect at lower retention times.

6.5 DRAM cache sensitivity

The above results assumed a ReRAM-only memory system. MORSE can also benefit hybrid memory systems, *e.g.* with a DRAM cache in front of the ReRAM. Although our simulator does not model DRAM caches, we increased the size and associativity of our CPU's LLC to study the traffic reduction effects. Figure 15 plots the effective SWA_{end} for LLC sizes from 64MB to 2GB with 64-way associativity.

As expected, larger LLCs (and hence, correspondingly larger DRAM caches) reduce the traffic to main memory which reduces the number of soft write opportunities, leading to lower effective SWA_{end}. In Figure 15, the SWA_{end} is 5.22 at 64MB, but it drops to 1.82 by 2GB. While the LLC capacities in Figure 15 are small for DRAM caches, our overall system and benchmark inputs are also very small. (Main memory is only 16 GB). For scaled up systems and benchmarks beyond the reach of simulation, in fact, we expect the smaller LLC capacities in Figure 15 to be more representative. For example, assuming a DRAM cache capacity 32x smaller than main memory [57], Figure 15 predicts the SWA_{end} will be 3.3 (the 256MB data point). Notice, the reduction of write traffic which lowers the soft write opportunity is itself a benefit. We find the combination of a DRAM cache with soft writes yields an aggregate endurance improvement of 6.6x for the 256MB capacity.

7 Related Work

Multi-level cell PCM exhibits a tradeoff between write latency and retention. Previous work leveraged this to improve performance at the expense of retention. Techniques include compiler-based approaches [36, 53, 56], NVM as a cache for SSDs [28], persistent programming frameworks [42], and architectural techniques [85, 86]. Of these, RRM [86] is the best-performing technique that (like our

work) targets systems with CPUs and non-volatile main memory. We adapted RRM to control soft writes for single-level cell ReRAM main memory to optimize energy and endurance. Our results show MORSE improves upon RRM significantly.

Mellow Writes [27, 84] improve ReRAM endurance by spreading the same (or slightly greater) write energy over a longer write cycle during idle periods in the memory system, thus minimally impacting contention and performance. Because a similar write energy is used, Mellow Writes do not impact the CF size, so retention remains roughly the same. While Mellow Writes only helps endurance, MORSE improves both endurance and energy since it employs "true" soft writes that dissipate less energy (though at the expense of reduced retention time and the need to refresh). Notice, MORSE and Mellow Writes are orthogonal. MORSE could make its soft or hard writes mellow, and further improve lifetime.

All of the above works try to reduce the impact of *individual writes* to NVM. Another line of work tries to reduce the *number of writes*. This includes methods that only write the modified bits [4, 7, 10, 18, 77], encode the data [12, 13, 24, 43, 48, 52, 70, 75, 89], compress the data [25, 51, 74], or utilize asymmetries in set and reset operations [7, 62, 77, 80]. These techniques are also orthogonal to our work and can be applied with soft writes.

Another write reduction technique is LLC management policies to reduce the writeback count [3, 55, 71, 82, 91]. These policies can only target writebacks with short overwrite times, leaving a lot of soft write opportunities. MORSE being agnostic to LLC management can adapt to these policies as long as the OTD gets the needed data.

8 Conclusion

This paper exploits the retention versus energy and endurance tradeoff in ReRAM. Our work identifies the relationship between overwrite time, retention time, and soft write advantage that governs the profitability of using soft writes, and makes the key observation that overwrite time is correlated to store PC. We develop an architectural technique, called MORSE, that learns the dominant overwrite time on a per-store PC basis, and uses the store locality information to predict a soft versus hard write decision at every dynamic store instruction. We show that MORSE improves endurance by 2.5x to 4.1x compared to a state-of-the-art soft write technique, RRM, using simulation-based study. MORSE also comes within 19.8% of the Oracle's endurance. Finally, we integrate MORSE with an existing wear leveling technique and show that it improves actual lifetime by 6.2x.

Acknowledgments

The authors would like to thank Martin Peckerar, Bruce Jacob, and Candace Walden for their valuable feedback, comments, and insights on Resistive RAM technology. This work was supported in part by the Semiconductor Research Corporation (SRC) through their AI Hardware program under contract #2020-AH-2937, and in part by the Department of Defense.

References

[1] Neha Agarawal and Thomas F. Wenisch. 2017. Thermostat: Application-Transparent Page Management for Two-Tiered Main Memory. In *Proceedings of the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems*.

[2] Mohammad Arjomand, Mahmut T Kandemir, Anand Sivasubramaniam, and Chita R Das. 2016. Boosting access parallelism to PCM-based main memory. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 695–706.

[3] Mohammad Bakhshalipour, Aydin Faraji, Seyed Armin Vakil Ghahani, Farid Samandi, Pejman Lotfi-Kamran, and Hamid Sarbazi-Azad. 2019. Reducing write-backs through in-cache displacement. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 24, 2 (2019), 1–21.

[4] Jie Chen, Ron C Chiang, H Howie Huang, and Guru Venkataramani. 2012. Energy-aware writes to non-volatile main memory. *ACM SIGOPS Operating Systems Review* 45, 3 (2012), 48–52.

[5] Ying-Chen Chen. 2019. *Selector-less resistive random access memory (RRAM) with intrinsic nonlinearity for crossbar array applications*. Ph. D. Dissertation.

[6] Zixuan Chen, Huaqiang Wu, Bin Gao, Dong Wu, Ning Deng, He Qian, Zhichao Lu, Brent Haukness, M Kellam, and Gary Bronner. 2017. Performance improvements by SL-current limiter and novel programming methods on 16MB RRAM chip. In *2017 IEEE International Memory Workshop (IMW)*. IEEE, 1–4.

[7] Sangyeun Cho and Hyunjin Lee. 2009. Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*. 347–357.

[8] Sergiu Clima, YY Chen, Andrea Fantini, Ludovic Goux, Robin Degraeve, Bogdan Govoreanu, Geoffrey Pourtois, and Malgorzata Jurczak. 2015. Intrinsic tailing of resistive states distributions in amorphous HfO x and TaO x based resistive random access memories. *IEEE Electron Device Letters* 36, 8 (2015), 769–771.

[9] Crossbar. 2020. Crossbar ReRAM Overview. (2020). https://www.crossbar-inc.com/technology/reram-overview/

[10] Wei Dong, Xin Li, Yanbin Li, Meikang Qiu, Lei Dou, Lei Ju, and Zhiping Jia. 2015. Minimizing update bits of NVM-based main memory using bit flipping and cyclic shifting. In *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*. IEEE, 290–295.

[11] Subramanya R Dulloor, Amitabha Roy, Zheguang Zhao, Narayanan Sundaram, Nadathur Satish, Rajesh Sankaran, Jeff Jackson, and Karsten Schwan. 2016. Data Tiering in Heterogeneous Memory Systems. In *Proceedings of the 2016 EuroSys Conference*.

[12] Ali Eslami, Alfredo Velasco, Alireza Vahid, Georgios Mappouras, Robert Calderbank, and Daniel J. Sorin. 2015. Writing without Disturb on Phase Change Memories by Integrating Coding and Layout Design *(MEMSYS '15)*. Association for Computing Machinery, New York, NY, USA, 71–77. https://doi.org/10.1145/2818950.2818962

[13] Dan Feng, Jie Xu, Yu Hua, Wei Tong, Jingning Liu, Chunyan Li, and Yiran Chen. 2019. A low-overhead encoding scheme to extend the lifetime of nonvolatile memories. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 10 (2019), 2516–2529.

[14] Alexandre P. Ferreira, Miao Zhou, Santiago Bock, Bruce Childers, Rami Melhem, and Daniel Mossé. 2010. Increasing PCM main memory lifetime. In *2010 Design, Automation Test in Europe Conference Exhibition (DATE 2010)*. 914–919. https://doi.org/10.1109/DATE.2010.5456923

[15] Cyril Goutte and Eric Gaussier. 2005. A probabilistic interpretation of precision, recall and F-score, with implication for evaluation. In *European conference on information retrieval*. Springer, 345–359.

[16] Alessandro Grossi, Elisa Vianello, Mohamed M Sabry, Marios Barlas, Laurent Grenouillet, Jean Coignus, Edith Beigne, Tony Wu, Binh Q Le, Mary K Wootters, et al. 2019. Resistive RAM endurance: Array-level characterization and correction techniques targeting deep learning applications. *IEEE Transactions on Electron Devices* 66, 3 (2019), 1281–1288.

[17] Tae Jun Ham, Bharath K Chelepalli, Neng Xue, and Benjamin C Lee. 2013. Disintegrated control for energy-efficient and heterogeneous memory systems. In *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 424–435.

[18] Andrew Hay, Karin Strauss, Timothy Sherwood, Gabriel H Loh, and Doug Burger. 2011. Preventing PCM banks from seizing too much power. In *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 186–195.

[19] Yukio Hayakawa, Atsushi Himeno, Ryutaro Yasuhara, W Boullart, E Vecchio, T Vandeweyer, T Witters, D Crotti, M Jurczak, S Fujii, et al. 2015. Highly reliable TaO x ReRAM with centralized filament for 28-nm embedded application. In *2015 Symposium on VLSI Technology (VLSI Technology)*. IEEE, T14–T15.

[20] Taekyung Heo, Yang Wang, Wei Cui, Jaehyuk Huh, and Lintao Zhang. 2020. Adaptive page migration policy with huge pages in tiered memory systems. *IEEE Trans. Comput.* 71, 1 (2020), 53–68.

[21] Takahiro Hirofuchi and Ryousei Takano. 2016. RAMinate: Hypervisor-based Virtualization for Hybrid Main Memory Systems. In *Proceedings of the ACM Symposium on Cloud Computing*. Santa Clara, CA.

[22] Jianming Huang, Yu Hua, Pengfei Zuo, Wen Zhou, and Fangting Huang. 2020. An Efficient Wear-Level Architecture Using Self-Adaptive Wear Leveling. In *49th International Conference on Parallel Processing - ICPP* (Edmonton, AB, Canada) *(ICPP '20)*. Association for Computing Machinery, New York, NY, USA, Article 11, 11 pages. https://doi.org/10.1145/3404397.3404405

[23] D. Ielmini, Federico Nardi, Carlo Cagli, and Andrea L. Lacaita. 2010. Trade-off Between Data Retention and Reset in NIO RRAMs. In *Proceedings of the IEEE International Reliability Physics Symposium*.

[24] Adam N Jacobvitz, Robert Calderbank, and Daniel J Sorin. 2013. Coset coding to extend the lifetime of memory. In *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 222–233.

[25] Amin Jadidi, Mohammad Arjomand, Mohammad Khavari Tavana, David R Kaeli, Mahmut T Kandemir, and Chita R Das. 2017. 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, 85–96.

[26] Meenatchi Jagasivamani. 2020. *Resistive RAM Based Main-Memory Systems: Understanding the Opportunities, Limitations, and Tradeoffs*. Ph. D. Dissertation. University of Maryland, College Park. https://drum.lib.umd.edu/handle/1903/26228

[27] Mohammad Reza Jokar, Lunkai Zhang, and Frederic T Chong. 2018. Cooperative nv-numa: prolonging non-volatile memory lifetime through bandwidth sharing. In *Proceedings of the International Symposium on Memory Systems*. 67–78.

[28] Dongwoo Kang, Seungjae Baek, Jongmoo Choi, Donghee Lee, Sam H Noh, and Onur Mutlu. 2015. Amnesic cache management for non-volatile memory. In *2015 31st Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 1–13.

[29] R.E. Kessler, E.J. McLellan, and D.A. Webb. 1998. The Alpha 21264 microprocessor architecture. In *Proceedings International Conference on Computer Design. VLSI in Computers and Processors (Cat. No.98CB36273)*. 90–95. https://doi.org/10.1109/ICCD.1998.727028

[30] Sunwoong Kim, Hyunmin Jung, Woojae Shin, Hyokeun Lee, and Hyuk-Jae Lee. 2019. HAD-TWL: Hot Address Detection-Based Wear Leveling for Phase-Change Memory Systems with Low Latency. *IEEE Computer Architecture Letters* 18, 2 (2019), 107–110. https://doi.org/10.1109/LCA.2019.2929393

[31] Jagadish B Kotra, Haibo Zhang, Alaa R Alameldeen, Chris Wilkerson, and Mahmut T Kandemir. 2018. Chameleon: A dynamically reconfigurable heterogeneous memory system. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 533–545.

[32] Corey Lammie, Mostafa Rahimi Azghadi, and Daniele Ielmini. 2021. Empirical Metal-Oxide RRAM Device Endurance and Retention Model for Deep Learning Simulations. *Semiconductor Science and Technology* 36 (2021).

[33] Albert Lee, Chieh-Pu Lo, Chien-Chen Lin, Wei-Hao Chen, Kuo-Hsiang Hsu, Zhibo Wang, Fang Su, Zhe Yuan, Qi Wei, Ya-Chin King, et al. 2017. A ReRAM-based nonvolatile flip-flop with self-write-termination scheme for frequent-OFF fast-wake-up nonvolatile processors. *IEEE Journal of Solid-State Circuits* 52, 8 (2017), 2194–2207.

[34] Benjamin C Lee, Engin Ipek, Onur Mutlu, and Doug Burger. 2009. Architecting phase change memory as a scalable dram alternative. In *Proceedings of the 36th annual international symposium on Computer architecture*. 2–13.

[35] Benjamin C Lee, Ping Zhou, Jun Yang, Youtao Zhang, Bo Zhao, Engin Ipek, Onur Mutlu, and Doug Burger. 2010. Phase-change technology and the future of main memory. *IEEE micro* 30, 1 (2010), 143–143.

[36] Qingan Li, Lei Jiang, Youtao Zhang, Yanxiang He, and Chun Jason Xue. 2013. Compiler directed write-mode selection for high performance low power volatile PCM. In *Proceedings of the 14th ACM SIGPLAN/SIGBED conference on Languages, compilers and tools for embedded systems*. 101–110.

[37] 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 *MICRO 42: Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*. 469–480.

[38] Sheng Li, Ke Chen, Jung Ho Ahn, Jay B. Brockman, and Norman P. Jouppi. [n. d.]. CACTI-P: Architecture-level modeling for SRAM-based structures with advanced leakage reduction techniques. In *ICCAD: International Conference on Computer-Aided Design*. 694–701.

[39] Duo Liu, Tianzheng Wang, Yi Wang, Zili Shao, Qingfeng Zhuge, and Edwin Sha. 2013. Curling-PCM: Application-specific wear leveling for phase change memory based embedded systems. In *2013 18th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 279–284. https://doi.org/10.1109/ASPDAC.2013.6509609

[40] Qingyue Liu and Peter Varman. 2017. Ouroboros wear leveling for NVRAM using hierarchical block migration. *ACM Transactions on Storage (TOS)* 13, 4 (2017), 1–31.

[41] Qingyue Liu and Peter Varman. 2017. Ouroboros Wear-Levling: A Two-Level Hierarchical Wear-Leveling Model for NVRAM. In *Proceedings of the International Conference on Massive Storage Systems and Technology*. Santa Clara, CA.

[42] Ren-Shuo Liu, De-Yu Shen, Chia-Lin Yang, Shun-Chih Yu, and Cheng-Yuan Michael Wang. 2014. NVM Duet: Unified working memory and persistent store architecture. *ACM SIGARCH Computer Architecture News* 42, 1 (2014), 455–470.

[43] Huizhang Luo, Liang Shi, Qiao Li, Chun Jason Xue, and Edwin H-M Sha. 2018. Energy, latency, and lifetime improvements in MLC NVM with enhanced WOM code. In *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 554–559.

[44] Kaisheng Ma, Xueqing Li, Karthik Swaminathan, Yang Zheng, Shuangchen Li, Yongpan Liu, Yuan Xie, John Jack Sampson, and Vijaykrishnan Narayanan. 2016. Nonvolatile processor architectures: Efficient, reliable progress with unstable power. *IEEE Micro* 36, 3 (2016), 72–83.

[45] Pierre Michaud, Andre Seznec, and Richard Uhlig. 1997. Trading Conflict and Capacity Aliasing in Conditional Branch Predictors. In *Proceedings of the 24th International Symposium on Computer Architecture*.

[46] C Nail, G Molas, P Blaise, G Piccolboni, B Sklenard, C Cagli, M Bernard, A Roule, M Azzaz, E Vianello, et al. 2016. Understanding RRAM endurance, retention and window margin trade-off using experimental results and simulations. In *2016 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 4–5.

[47] Prashant J Nair, Dae-Hyun Kim, and Moinuddin K Qureshi. 2013. ArchShield: Architectural framework for assisting DRAM scaling by tolerating high error rates. *ACM SIGARCH Computer Architecture News* 41, 3 (2013), 72–83.

[48] Kazuteru Namba and Fabrizio Lombardi. 2016. A coding scheme for write time improvement of phase change memory (PCM) systems. *IEEE Transactions on Multi-Scale Computing Systems* 2, 4 (2016), 291–296.

[49] Federico Nardi, Daniele Ielmini, Carlo Cagli, S Spiga, M Fanciulli, Ludovic Goux, and DJ Wouters. 2011. Control of filament size and reduction of reset current below 10 μA in NiO resistance switching memories. *Solid-State Electronics* 58, 1 (2011), 42–47.

[50] Dimin Niu, Cong Xu, Naveen Muralimanohar, Norman P Jouppi, and Yuan Xie. 2013. Design of cross-point metal-oxide ReRAM emphasizing reliability and cost. In *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 17–23.

[51] Poovaiah M. Palangappa and Kartik Mohanram. 2016. 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)*. 90–101. https://doi.org/10.1109/HPCA.2016.7446056

[52] Poovaiah M Palangappa and Kartik Mohanram. 2018. RAPID: read acceleration for improved performance and endurance in MLC/TLC NVMs. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–7.

[53] Chen Pan, Mimi Xie, Jingtong Hu, Yiran Chen, and Chengmo Yang. 2014. 3M-PCM: Exploiting multiple write modes MLC phase change main memory in embedded systems. In *Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis*. 1–10.

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

[55] Bahareh Pourshirazi, Majed Valad Beigi, Zhichun Zhu, and Gokhan Memik. 2018. WALL: A writeback-aware LLC management for PCM-based main memory systems. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 449–454.

[56] Keni Qiu, Qingan Li, and Chun Jason Xue. 2014. Write mode aware loop tiling for high performance low power volatile PCM. In *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[57] 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 *2009 42nd Annual IEEE/ACM international symposium on microarchitecture (MICRO)*. IEEE, 14–23.

[58] Moinuddin K. Qureshi, Vijayalakshmi Srinivasan, and Jude A. Rivers. 2009. Scalable High Performance Main Memory System Using Phase-Change Memory Technology. In *Proceedings of the International Symposium on Computer ARchitecture*. Austin, TX.

[59] Daniel Sanchez and Christos Kozyrakis. 2013. ZSim: Fast and accurate microarchitectural simulation of thousand-core systems. *ACM SIGARCH Computer architecture news* 41, 3 (2013), 475–486.

[60] Nak Hee Seong, Dong Hyuk Woo, and Hsien-Hsin S Lee. 2010. Security refresh: Prevent malicious wear-out and increase durability for phase-change memory with dynamically randomized address mapping. *ACM SIGARCH computer architecture news* 38, 3 (2010), 383–394.

[61] Andre Seznec. 2010. A phase change memory as a secure main memory. *IEEE Computer Architecture Letters* 9, 1 (2010), 5–8.

[62] Shihao Song, Anup Das, Onur Mutlu, and Nagarajan Kandasamy. 2020. Improving phase change memory performance with data content aware access. In *Proceedings of the 2020 ACM SIGPLAN International Symposium on Memory Management*. 30–47.

[63] spec 2017. SPEC CPU2017 Benchmarks. . (2017). https://www.spec.org/cpu2017/

[64] Dmitrii Ustiugov, Mark Suterhland, Alexandros Daglis, Edouard Bugnion, Dionisios Pnevmatikatos, Javier Picorel, and Babak Falsafi. 2018. Design Guidelines for High-Performance SCM Hierarchies. In *Proceedings of the 4th International Symposium on Memory Systems*. National Harbor, D.C.

[65] Majed Valad Beigi, Bahareh Pourshirazi, Gokhan Memik, and Zhichun Zhu. 2020. DeepSwapper: A Deep Learning Based Page Swap Management Scheme for Hybrid Memory Systems. In *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*. 353–354.

[66] Ilia Valov, Rainer Waser, John R Jameson, and Michael N Kozicki. 2011. Electrochemical metallization memories—fundamentals, applications, prospects. *Nanotechnology* 22, 25 (2011), 254003.

[67] Candace Walden, Devesh Singh, Meenatchi Jagasivamani, Shang Li, Luyi Kang, Mehdi Asnaashari, Sylvain Dubois, Bruce Jacob, and Donald Yeung. 2021. Monolithically Integrating Non-Volatile Main Memory over the Last-Level Cache. *ACM Transactions on Architecture and Code Optimization (TACO)* 18, 4 (2021), 1–26.

[68] Guan Wang, Fei Peng, Lei Ju, Lei Zhang, and Zhiping Jia. 2014. Double circulation wear leveling for PCM-based embedded systems. In *Advanced Computer Architecture*. Springer, 190–200.

[69] Hao Wang, Jie Zhang, Sharmila Shridhar, Gieseo Park, Myoungsoo Jung, and Nam Sung Kim. 2016. DUANG: Fast and lightweight page migration in asymmetric memory systems. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 481–493.

[70] Jue Wang, Xiangyu Dong, Guangyu Sun, Dimin Niu, and Yuan Xie. 2011. Energy-efficient multi-level cell phase-change memory system with data encoding. In *2011 IEEE 29th International Conference on Computer Design (ICCD)*. IEEE, 175–182.

[71] Zhe Wang, Shuchang Shan, Ting Cao, Junli Gu, Yi Xu, Shuai Mu, Yuan Xie, and Daniel A Jiménez. 2013. WADE: Writeback-aware dynamic cache management for NVM-based main memory system. *ACM Transactions on Architecture and Code Optimization (TACO)* 10, 4 (2013), 1–21.

[72] Cong Xu, Dimin Niu, Naveen Muralimanohar, Rajeev Balasubramonian, Tao Zhang, Shimeng Yu, and Yuan Xie. 2015. Overcoming the challenges of crossbar resistive memory architectures. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 476–488.

[73] Cong Xu, Dimin Niu, Naveen Muralimanohar, Norman P Jouppi, and Yuan Xie. 2013. Understanding the trade-offs in multi-level cell ReRAM memory design. In *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[74] Jie Xu, Dan Feng, Yu Hua, Wei Tong, Jingning Liu, and Chunyan Li. 2018. Extending the lifetime of NVMs with compression. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1604–1609.

[75] Jie Xu, Dan Feng, Yu Hua, Wei Tong, Jingning Liu, Chunyan Li, and Wen Zhou. 2017. Improving performance of TLC RRAM with compression-ratio-aware data encoding. In *2017 IEEE International Conference on Computer Design (ICCD)*. IEEE, 573–580.

[76] Dongliang Xue, Chao Li, Linpeng Huang, Chentao Wu, and Tianyou Li. 2018. Adaptive memory fusion: Towards transparent, agile integration of persistent memory. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 324–335.

[77] Byung-Do Yang, Jae-Eun Lee, Jang-Su Kim, Junghyun Cho, Seung-Yun Lee, and Byoung-Gon Yu. 2007. A low power phase-change random access memory using a data-comparison write scheme. In *2007 IEEE International Symposium on Circuits and Systems*. IEEE, 3014–3017.

[78] Hongliang Yu and Yuyang Du. 2014. Increasing Endurance and Security of Phase-Change Memory with Multi-Way Wear-Leveling. *IEEE Trans. Comput.* 63, 5 (2014), 1157–1168. https://doi.org/10.1109/TC.2012.292

[79] Shimeng Yu, Yang Yin Chen, Ximeng Guan, and H.-S. Philip Wong. 2012. A Monte Carlo Study of the Low Resistance State Retention of HfOx Based Resistive Switching Memory. *Applied Physics Letters* (2012).

[80] Jianhui Yue and Yifeng Zhu. 2013. Accelerating write by exploiting PCM asymmetries. In *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 282–293.

[81] Furqan Zahoor, Tun Zainal Azni Zulkifli, and Farooq Ahmad Khanday. 2020. Resistive random access memory (RRAM): an overview of materials, switching mechanism, performance, multilevel cell (MLC) storage, modeling, and applications. *Nanoscale research letters* 15, 1 (2020), 1–26.

[82] Deshan Zhang, Lei Ju, Mengying Zhao, Xiang Gao, and Zhiping Jia. 2016. Write-back aware shared last-level cache management for hybrid main memory. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[83] Jialiang Zhang, Nicholas Beckwith, and Jing Jane Li. 2021. GORDON: Benchmarking Optane DC Persistent Memory Modules on FPGAs. In *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 97–105. https://doi.org/10.1109/FCCM51124.2021.00019

[84] Lunkai Zhang, Brian Neely, Diana Franklin, Dmitri Strukov, Yuan Xie, and Frederic T Chong. 2016. Mellow writes: Extending lifetime in resistive memories through selective slow write backs. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 519–531.

[85] Mingzhe Zhang, Lunkai Zhang, Lei Jiang, Frederic T Chong, and Zhiyong Liu. 2019. Quick-and-Dirty: An Architecture for High-Performance Temporary Short Writes in MLC PCM. *IEEE Trans. Comput.* 68, 9 (2019), 1365–1375.

[86] Mingzhe Zhang, Lunkai Zhang, Lei Jiang, Zhiyong Liu, and Frederic T. Chong. 2017. Balancing Performance and Lifetime of MLC PCM by Using a Region Retention Monitor. In *International Symposium on High Performance Computer Architecture*.

[87] Wangyuan Zhang and Tao Li. 2009. Exploring phase change memory and 3D die-stacking for power/thermal friendly, fast and durable memory architectures. In *2009 18th International Conference on Parallel Architectures and Compilation Techniques*. IEEE, 101–112.

[88] Yang Zhang, Dan Feng, Wei Tong, Jingning Liu, Chengning Wang, and Jie Xu. 2019. Tiered-ReRAM: A low latency and energy efficient TLC crossbar ReRAM architecture. In *2019 35th Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 92–102.

[89] Yang Zhang, Dan Feng, Wei Tong, Jingning Liu, Chengning Wang, and Jie Xu. 2019. Tiered-ReRAM: A Low Latency and Energy Efficient TLC Crossbar ReRAM Architecture. In *2019 35th Symposium on Mass Storage Systems and Technologies (MSST)*. 92–102. https://doi.org/10.1109/MSST.2019.00-13

[90] Meiran Zhao, Huaqiang Wu, Bin Gao, Xiaoyu Sun, Yuyi Liu, Peng Yao, Yue Xi, Xinyi Li, Qingtian Zhang, Kanwen Wang, et al. 2018. Characterizing endurance degradation of incremental switching in analog RRAM for neuromorphic systems. In *2018 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 20–2.

[91] Miao Zhou, Yu Du, Bruce Childers, Rami Melhem, and Daniel Mossé. 2012. Writeback-aware partitioning and replacement for last-level caches in phase change main memory systems. *ACM Transactions on Architecture and Code Optimization (TACO)* 8, 4 (2012), 1–21.

[92] Ping Zhou, Bo Zhao, Jun Yang, and Youtao Zhang. 2009. A durable and energy efficient main memory using phase change memory technology. *ACM SIGARCH computer architecture news* 37, 3 (2009), 14–23.