SRTP: Predicting Store Reuse Time to Improve ReRAM Energy and Endurance

Devesh Singh, Donald Yeung

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 studies demonstrate 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.

Whether or not soft writes provide a benefit depends on the amount of time between back-to-back writes to the same data, which we call the *store reuse time*. As long as the cost for the soft write and its refreshes within the store reuse time window is less than the cost for a hard write, then the original soft write is profitable. Otherwise, it would have been better to perform a hard write in order to eliminate the refreshes.

We propose SRTP, a predictor that learns the store reuse times between back-to-back writes to main memory, and associates them with static store instructions in a prediction table. As dynamic stores execute, SRTP predicts whether a soft or hard write is best based on the magnitude of the predicted store reuse time. 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 SRTP provides 2.6x - 4.1x improvement in endurance and 2.9x - 4.2x improvement in write energy over a state-of-the-art predictor. We also show that SRTP is within 18.5% of the Oracle policy. Finally, we integrate SRTP with a prior wear leveling technique, called Ouroboros, and show that SRTP improves actual memory system lifetime by 5.0x over a baseline that only performs hard writes.

1. INTRODUCTION

Emerging non-volatile memory technologies, such as resistive random access memory (ReRAM) and phase change memory (PCM), offer much higher memory capacities compared to DRAM. At the same time, they also provide improved power efficiency since their non-volatility eliminates the need for refresh operations. Given these benefits, not only have non-volatile memories been studied extensively in research [2, 39, 40, 50, 66, 84, 88, 99, 104], but they have also become commercially available (*e.g.*, Intel's Optane memory [25, 27]), leading to their deployment in real systems.

The main disadvantage of non-volatile memories, however, is the high cost of their writes. Not only do their write operations incur greater latency than read operations, but they also consume significantly more energy and cause the memory system to 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 along with the non-volatile memory [1, 13, 17, 21, 22, 66, 74, 75, 79]. For example, the DRAM can be placed in front of the non-volatile memory, acting as a cache that automatically filters the write stream [66].

Rather than reduce the write frequency, another approach for addressing the high cost of writes in non-volatile memories is to give up 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 (either 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 [10]). However, in main memory, years-long retention is overkill since the written data will be over-written anyways 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 example, a soft write to ReRAM forms a smaller conductive filament (CF) within the metal oxide layer which is susceptible to dissolution [24, 37, 91], 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, a large number of refreshes are needed in between writes to infrequently updated data, which defeats the purpose 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.

Recently, researchers proposed the Region Retention Monitor (RRM) [98] for controlling the selection of soft versus hard writes in MLC PCM memory systems. Unlike our work which tries to improve energy and endurance, RRM uses soft writes to boost performance by reducing the number of write cycles, and hence the latency, of soft writes compared to hard writes. RRM tracks write frequency at the page level, and uses soft writes only for the most frequently written pages. These "hot pages" are kept in a hardware table which issues refresh operations to the softly written data. For all other pages (not found in the table), RRM performs hard writes instead, thus eliminating many harmful refresh operations.

Although RRM was originally designed for performance, we adapted it to also improve energy and endurance. Unfortunately, we find RRM does not translate well to these other objective functions. The problem is RRM performs soft / hard write selection using an ad hoc heuristic which is oblivious to the actual relationship between store locality and retention time that governs the effectiveness of soft writes.

Specifically, following every soft write, refresh operations are needed periodically in intervals equal to the retention time, as shown in Figure 1. Whether or not the soft writes provide a benefit 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). In our work, we associate this reuse time back to the last store instruction that caused the initial writeback, so we refer to it as the *last-touch store reuse time*, or simply the *store reuse time*. As long as the cost for the soft writes occurring within this time window is less than the cost for a hard write, then the original soft write is profitable. Otherwise, it would have been better to perform a hard write in order to eliminate the refreshes. For instance, if one considers write energy as the cost metric, then soft writes should be performed whenever the following inequality holds:

$$\frac{StoreReuseTime}{RetentionTime} < \frac{E_{HdWr}}{E_{SfWr}} \qquad (1)$$

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 energy in more detail.)

To achieve the greatest gains, a separate selection decision should be made at *every* dynamic store instruction. But RRM is ineffective at such fine-grain control because it aggregates memory-side access information at a coarse page granularity. Our work makes the key observation that *store reuse times are correlated to store PCs*. Although store locality can vary significantly across different memory accesses, the accesses performed by the same static store instruction tend to exhibit a single dominant store reuse time. By training a CPU-side predictor on the dominant reuse time for each static store, Inequality 1 can be predicted accurately for every dynamic write to main memory. We call this predictor the *Store Reuse Time Predictor, or SRTP*.

Numerous prior techniques have leveraged locality information for optimizing systems. In particular, many examples exist in the literature, including cache and TLB replacement policies [11, 30, 33, 48, 52, 69, 83], cache partitioning techniques [4, 35, 65] or tracking reuse time for DRAM rows [18, 81]. However, our work differs in that the reuse times we try to predict are much, much longer. Previous techniques mainly studied cache reuse, so the reuse times involved were relatively short–on the order of milliseconds. In contrast, SRTP predicts reuse potentially spanning multiple soft write retention intervals, which can last *several seconds*.

Training a predictor on multi-second events could result in long training times. Moreover, infeasibly large predictors might be necessary to track the huge number of events that occur over such long time horizons. Fortunately, store reuse tends to be stable on a per-store PC basis, as mentioned earlier. Also, a small number of static store instructions typically accounts for a large fraction of the dynamic stores executed.

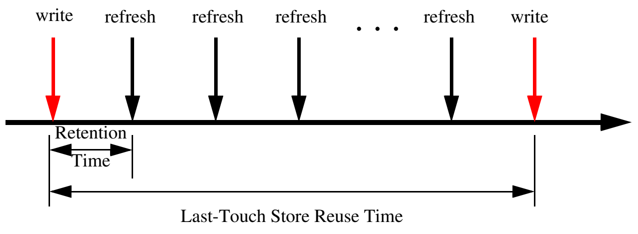

Figure 1: The profitability of soft writes depends on the last-touch store reuse time, and hence, the number of refreshes needed before the next write.

SRTP employs a small buffer, called the *Reuse Time Detector (RTD)*, tracking only two dynamic instances per static store instruction. The RTD can learn the most important store reuse times even when they last for a second or more.

Besides predicting Inequality 1, our technique must also issue refreshes to the softly written data. Unlike our prediction mechanism which resides on the CPU side, we track refresh operations on the memory side at the page level, similar to RRM. One difference, however, is that our refresh tracking information is much larger because SRTP capitalizes on many more soft write opportunities. Whereas RRM combines the prediction and refresh information in the same table, we decouple them and move the refresh information–which is much larger but accessed less frequently–to main memory.

Our work also integrates SRTP with wear leveling. We modify an existing wear leveling technique, Ouroboros [46], to account for the asymmetric wear across soft and hard writes. Advanced wear leveling techniques like Ouroboros already store per-page address translation tables in main memory. We integrate SRTP's refresh tracking information into these existing data structures. Although SRTP's refresh information is large, it only adds 3.8% more memory on top of Ouroboros' address translation tables.

Our work makes the following contributions:

- We adapt RRM, which was originally designed for performance, to control soft / hard write selection for improving energy and endurance in ReRAM-based memory systems.
- We present the Oracle algorithm for selecting soft versus hard writes, and show that RRM leaves a lot of room for improvement.
- We propose a novel hardware predictor, called *SRTP*, to select soft versus hard writes at the CPU side on a per-dynamic store instruction basis.
- We integrate SRTP with Ouroboros wear leveling.
- We undertake a simulation-based evaluation of SRTP, and show that it beats RRM by 410% and comes within 18.5% of the Oracle. We also demonstrate that SRTP can improve lifetime by 5.0x when coupled with practical wear leveling.

The rest of this paper is organized as follows. Section 2 presents background on retention versus energy and endurance tradeoffs. Then, Section 3 provides motivation, and Section 4 describes our SRTP technique. Next, Section 5 discusses experimental methodology, and Section 6 presents the results. Finally, Section 7 discusses related work, and Section 8 concludes the paper.

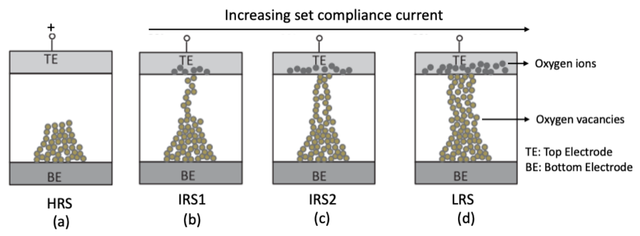

Figure 2: Switching in a ReRAM bitcell (from [85]).

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 the oxygen vacancies [91]. The time it takes to lose the stored data depends on the size, or cross-sectional area, of the CF: the wider the CF, the longer the data lasts, whereas the narrower the CF, the sooner the data is lost [76, 91, 93].

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 need to use a reset current, I_{RESET}, that is also linearly proportional to CF_A in order to dissolve the formed CF [56]. 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 [29, 53, 102]. So, endurance is also inversely proportional to CF_A. In other words, we have the following relationship:

$$CF_A \propto Write\ Energy \propto Endurance^{-1} \tag{2}$$

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 [56].

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 [37], leading to the following relationship:

$$RetentionTime \propto e^{p*CF_D} \tag{3}$$

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 [9, 91]. The retention time for bitcells at the tail of this distribution can be further reduced by up to 100x [91]. In our work, we use the model from [37] to derive retention time for a given CF size, and then further reduce it by 2 orders of magnitude to take reliability of the tail bits into account. This brings the error rate into the range that can be handled by SECDED Error Correction Codes(ECC) [54]. Simple hamming codes based ECC implementation that only corrects one bit already has 12.5% storage overhead for 64 bit words [26]. Increasing the error correction capability is not an option as it will increase both storage and computational overhead by the factor of correctable bits [73]. Byte addressable main memories cannot amortize the storage overhead over large blocks and expensive ECC calculations are difficult to implement at memory bus speeds resulting into significant area and performance penalties [26].

In particular, our work employs a soft write that incurs 10x less energy compared to the 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 100x 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 [10]. So, the retention time for soft writes goes down to 10 seconds. This represents a worst-case retention time (due to the 100x for the tail bits), making our energy and endurance results conservative.

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 that soft writes 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 objective. Inequality 1 shows the SWA for optimizing endurance–*i.e.*, SWA_{end}. Because endurance is proportional to write energy (Expression 2), SWA_{end} is just the ratio of the hard and soft write energies, which is 10x as discussed above.

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 before performing a soft write (which doesn't affect endurance but does affect energy). 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:

$$E_{HdWr} = E_{SfWr} + (E_{SfWr} + E_{Read})(SWA_{egy} - 1)$$

$$SWA_{egy} = \frac{E_{HdWr} + E_{Read}}{E_{SfWr} + E_{Read}} \tag{4}$$

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

$$\frac{StoreReuseTime}{RetentionTime} < \frac{E_{HdWr} + E_{Read}}{E_{SfWr} + E_{Read}} \tag{5}$$

3. MOTIVATION

This section motivates our work by quantifying how much room exists for soft write techniques to improve energy and endurance. We consider a state-of-the-art soft write selection

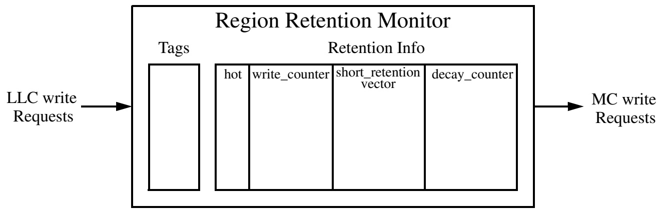

Figure 3: Region Retention Monitor hardware table. (Adapted from [98]).

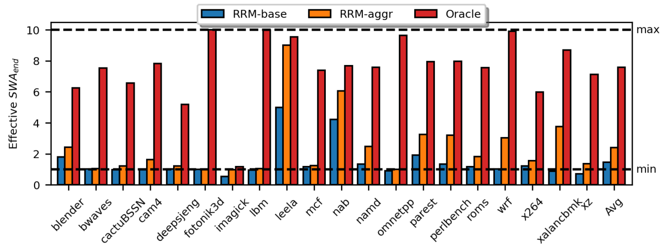

Figure 4: Effective SWA_{end} for RRM and Oracle assuming 10 second retention time and 1/10th wear for soft writes.

technique, RRM, adapting it for write energy and endurance gains. Then, we present the Oracle policy that provides the upper bound and compare RRM against it. Finally, we discuss insights into how we can close the gap between them.

3.1 Region Retention Monitor

In general, soft writes should be employed for frequently written data, whereas hard writes should be used for infrequently written data. A natural strategy, therefore, is to profile the write frequency, and to use the profile for driving soft / hard write decisions. The Region Retention Monitor (RRM) [98] is a recent soft write technique that adopts this approach.

Figure 3 illustrates RRM. RRM employs a hardware table between the LLC and memory controller (MC). The RRM table observes writes to dirty cache blocks in the LLC and makes a soft write decision on evictions as they write back to an NVM main memory. Each entry in the table tracks information per page. A "write_counter" for a page is incremented whenever an LLC write is issued to the page, and a "hot" bit is set after a threshold number of LLC writes have been observed. Initially, writebacks use hard writes, but once a page's hot bit is set, writebacks switch to using soft writes.

The RRM table not only controls soft / hard write decisions, but it is also responsible for issuing refresh operations. Each entry in the RRM table with the hot bit set periodically issues refresh operations to its corresponding page. RRM tracks the memory blocks within a page that have been softly written using a bit vector, called the "short_retention_vector," and only refreshes those memory blocks marked in the bit vector.

After identifying the hot data for soft write candidates, it is also necessary to detect if and when the hot data transition back to being infrequently written; otherwise, the refresh operations could nullify the benefits of the soft writes. In RRM, this is implemented using the "decay_counter." Periodically, the decay_counter is incremented until it rolls over, at which point the write_counter is re-checked to see if it still meets the threshold for a hot page. If so, the write_counter is reduced in half and the decay_counter is reset. If the page stops receiving writebacks, then the write_counter will eventually drop below the hot threshold. When this happens, the hot bit for the page is cleared and hard writes are issued to all softly written data. This is known as a *reset hard write*, which persists the data and allows the RRM table to stop issuing refreshes to the page.

3.2 Oracle Soft Write Selection

To assess the effectiveness of RRM, we developed the Oracle policy for selecting soft versus hard writes. As shown in Figure 1, the best soft / hard write decision depends on how far apart back-to-back writes to the same memory location are. In our experiments, we simulate an oracle mode which records the timestamp for every LLC writeback so that this reuse time can be computed precisely. More specifically, when performing a writeback, the simulator looks up the timestamp for the last writeback to the same memory block, and computes the reuse time. The simulator then uses this reuse time to evaluate Inequality 1 and determine whether the *previous* writeback should have used a soft or hard write. (The Oracle currently optimizes for endurance; we could also evaluate Inequality 5 to optimize for total energy). Accordingly, the simulator attributes (after the fact) the write energy for either a soft or hard write to the previous writeback. In the case of a soft write, the simulator also attributes the write energy for $\frac{ReuseTime}{RetentionTime}$ number of refresh operations as well. Then, the simulator updates the memory block's timestamp with the current time, and continues execution. We implemented this Oracle policy and compare RRM against it. (Section 5 will provide more details on our simulator).

Figure 4 shows the endurance results achieved across several SPEC CPU2017 benchmarks [72]. In Figure 4, we plot the total write energy for the soft write technique (either RRM or Oracle) normalized to the total write energy when all writes are hard writes. We call this the *effective SWA_{end}*. This metric quantifies the improvement in wear out provided by the soft write technique. (Given the parameters from Section 2, the maximum effective SWA_{end} is 10x, which occurs when every write is a soft write, and there are no refreshes). For RRM, two sets of bars are reported: "RRM-base" which employs an RRM table with 4096 entries, matching the original RRM paper [98], and "RRM-aggr" which employs a more aggressive 32,768-entry table.

As Figure 4 shows, there is significant room for improving RRM. RRM-base has an effective SWA_{end} of just 1.5x. One reason why these gains are so modest is that RRM-base can only track 4096 hot pages, which is not enough for our SPEC 2017 benchmarks. When the RRM table is increased to 32,768 entries–*i.e.*, using RRM-aggr–the effective SWA_{end} jumps to 2.4x. However, Figure 4 shows the Oracle policy achieves an effective SWA_{end} of 7.6x. Not only is this fully 5x better than RRM-base, but it is still over 3x better than RRM-aggr. This motivates the need to bridge the gap between prior techniques and what is theoretically possible.

3.3 CPU-Side Reuse Time Prediction

The Oracle policy from Section 3.2 knows the future reuse time for every LLC writeback, and uses it to evaluate Inequality 1 at each dynamic write. Our work tries to emulate this

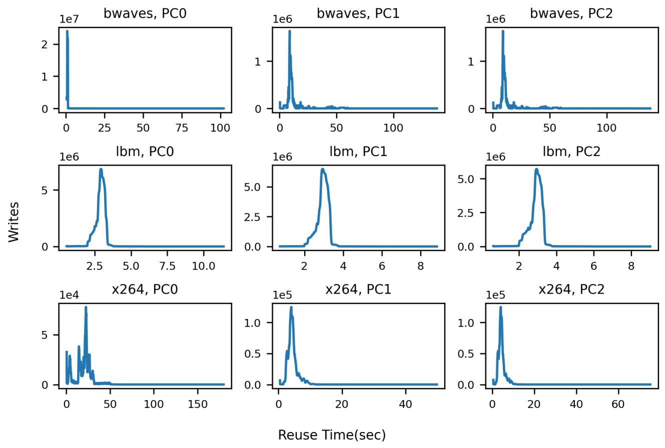

Figure 5: Reuse time histograms for the top three last-touch stores from three SPEC CPU2017 benchmarks.

fine-grained soft write selection performed by the Oracle. In order to do so, we observe the future reuse time for certain dynamic writes (which the Oracle omnisciently knows for all writes) and use these samples to *predict* the soft write criterion (either Inequality 1 or Inequality 5) for every dynamic store. Our technique is called the Store Reuse Time Predictor, or SRTP.

Unfortunately, as shown in Figure 4, RRM is not effective at such predictions. The problem is that RRM keeps memory access information at the memory side. In order to make decisions like the Oracle, RRM would have to track access counts on a per-memory location basis which is completely out of the question. RRM instead aggregates access counts at the page level, but even so, the amount of predictor state can still be very large. This not only results in large predictors, but it also incurs long training times (training is needed for *every page* in memory) which reduces the number of soft write opportunities that prediction can capitalize upon.

Rather than make predictions from the memory side, SRTP performs the predictions from the CPU side instead. It observes the reuse times between writes to main memory, and uses them to directly assess either Inequality 1 or 5. *SRTP associates the resulting soft write decisions back to the static store instructions linked to those writes.* As the store instructions execute again, SRTP predicts for every dynamic instance either soft or hard write using the soft write decision previously made for the corresponding static store. Because there are a much smaller number of static store instructions than there are memory locations stored to, SRTP's predictor state is significantly reduced compared to RRM.

We find that CPU-side prediction can be very accurate thanks to *store correlation*. While reuse times can vary significantly across different dynamic writes to main memory, those linked to the same static store instruction tend to exhibit a single dominant reuse time. To illustrate, Figure 5 shows the reuse time histograms associated with the top-three static store instructions (PC0, PC1, and PC2) across three different SPEC CPU2017 benchmarks (bwaves, lbm, and x264). Reuse time correlation around a "peak" is clearly visible in each histogram. While the peak can be wide for some benchmarks (lbm) or accompanied by secondary peaks in other benchmarks (x264, PC0), there is nevertheless a single dominant reuse time in all cases. Hence, there is also a single soft write decision dictated by Inequality 1 or Inequality 5 that is appropriate for each static store instruction as well. By training our predictor per store PC, we can exploit this correlation to achieve high prediction accuracy.

Unlike prediction, which SRTP performs at the CPU side, refresh and reset hard writes are tied to specific memory locations, and must be performed at the memory side. Worse yet, as we approach the Oracle's performance, the amount of softly written data will increase, making the state for tracking refreshes and resets significantly larger. Fortunately, this state does not need to be accessed frequently. SRTP decouples the refresh and reset information from its predictor, and moves it into main memory to take advantage of higher capacities. Later, we show how this information can be integrated into wear leveling data structures.

4. STORE REUSE TIME PREDICTOR

Having motivated potential benefits, we now present the details of our SRTP technique. SRTP adds new hardware components to a multicore CPU. First, the Reuse Time Detector (RTD) learns the best soft write decision for each static store instruction by observing store reuse times. Second, the soft-write predictor (SWP) uses these learned soft write decisions to predict either soft or hard write for every dynamic store instruction. The SWP is replicated per core, with each module predicting for its associated core. Figure 6(a) shows how the RTD and SWP are integrated into a multicore CPU. Finally, SRTP maintains information for driving refresh and reset hard writes in main memory, which can be integrated with wear leveling. The rest of this section presents the details.

4.1 RTD

The RTD module, illustrated in Figure 6(b), is an associative buffer that performs training for the SWP module by observing reuse times between LLC writebacks. When a dirty LLC cache block writes back to main memory, a lookup is performed in the RTD. If no matching entry is found, the writeback block address (or tag) is filled into the RTD as long as a free entry exists (label ① in Figure 6(b)). Also, the time at which the writeback happened is filled into the RTD along with its tag (label ② in Figure 6(b)). Later on, if the same memory block is written back again, a match in the RTD will occur, 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 writeback-to-writeback reuse time can then be used to evaluate either Inequality 1 or Inequality 5, thus determining whether the original writeback should have used a soft or hard write.

In addition to the memory block address, each RTD entry is also tagged with the PC of the store instruction associated with the writeback (label ③ in Figure 6(b)). To enable this, we extend a few of the cache tags in the L1, L2, and LLC with a store PC field (label ④ in Figure 6(d)). Each time a store hit occurs to a tracked cache block in the L1, we write the instruction's PC into this field. Multiple store hits to the same L1 block simply overwrite each other's store PC. As dirty L1 cache blocks are evicted, they transfer their store PC to the L2 and then to the LLC. Thus, on an LLC writeback, the store PC used to access / fill the RTD represents

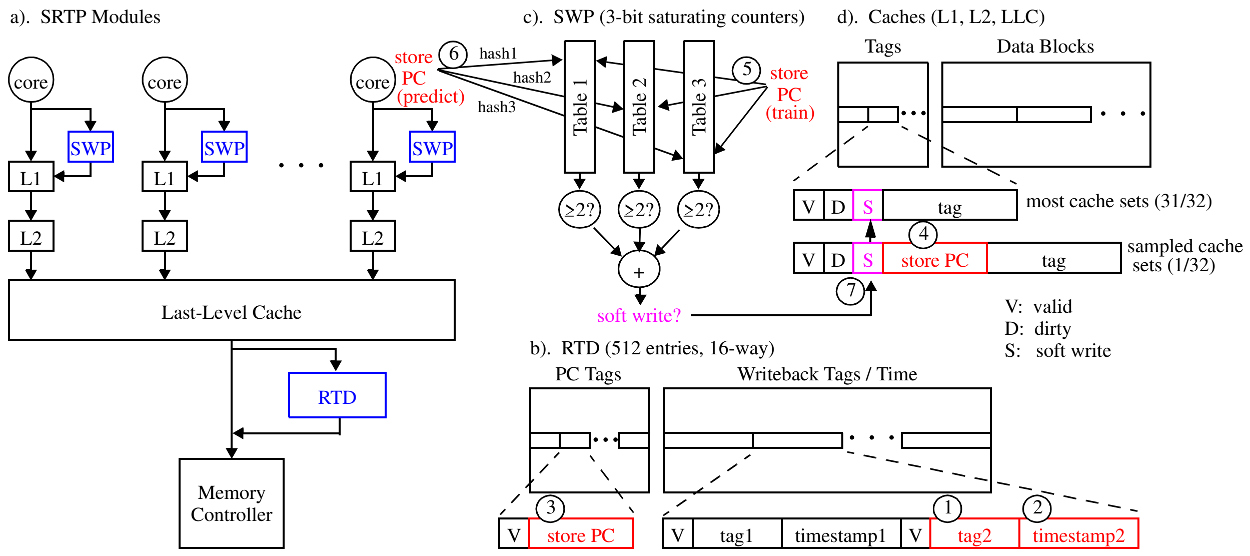

Figure 6: The Store Reuse Time Predictor (SRTP) (a) adds two hardware modules (in blue) to a multicore CPU: (b) a Reuse Time Detector (RTD) and (c) a per-core Soft Write Predictor (SWP). (d) SRTP also extends the CPU's caches with soft-write bits and store PC fields.

the last store instruction to hit the cache block prior to the writeback–*i.e.*, *the last-touch store*. (Notice, with both store PCs and memory blocks as tags, an RTD lookup involves two matches: the store PC is used to both index the RTD 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). To handle aliasing for multi-programmed workloads the store instruction tag could be extended with core-id bits. For an average SPEC CPU2017 benchmarks, 76 top last-touch store instructions are associated with over 90% of writebacks. Given the small number of important static last-touch stores and the large number of overall instruction addresses available, the chances any of these last-touch store instructions have same PC values for different workloads are minuscule. This allows us to omit the core-id bits without worrying about aliasing. The experiments simulate aliasing and the results for SRTP in Section 6 are pretty close to the oracle.

Because the last-touch store PC is associated with each writeback's timestamp, once a match occurs in the RTD and a reuse time is obtained, we can train the CPU-side (*i.e.*, PC-based) SWP module by associating the resulting soft-write decision computed from Inequality 1 or 5 with the store PC. This is indicated by label ⑤ in Figure 6(c). In Section 4.2, we will discuss this training procedure in more detail.

One reason why soft write decisions are so challenging to learn is because writeback-to-writeback reuse times can be extremely long, on the order of multiple seconds. In that length of time, a huge number of dynamic store instructions forming a huge number of writeback reuses will occur. Fortunately, the RTD module only needs to track a tiny fraction of these events for a couple of 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 RTD only needs to track a few store PCs. For the SPEC CPU2017 benchmarks, we find that 512 store PCs (*i.e.*, RTD entries) are sufficient to capture the "working set" of static store instructions responsible for most last-touch stores. To minimize conflicts, we employ 16-way set associativity in the RTD. (We use an LRU eviction policy within each set).

Second, as discussed in Section 3.3, most last-touch store PCs exhibit a single dominant reuse time thanks to store correlation. Thus, the RTD only needs to sample a small number of writeback-to-writeback reuse events. In our RTD implementation, we track just two memory blocks per store PC, as shown in Figure 6(b), permitting detection of only two reuses at a time. In addition, because we track so few memory blocks in the RTD, we only need a fraction of the cache blocks to carry a store PC field (label ④ in Figure 6(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 RTD entries and cache set sampling are sufficient to observe the dominant writeback-to-writeback reuse time for most store PCs.

Due to the long duration of writeback-to-writeback reuse events, the memory block tags must be allowed to linger in the RTD for a long time. Hence, *we never evict memory block tags*. Instead, the tags are removed when a matching lookup occurs and we complete a reuse time calculation. We also detect when a tag's timestamp expires–*i.e.*, the timestamp becomes so old that Inequality 1 or Inequality 5 (whichever one is being used) is guaranteed to be false even though we have yet to see a reuse occur. In that case, we conclude that hard write is the correct decision and eagerly remove the tag, saving the need to wait for the time remaining before the writeback reuse actually occurs.

4.2 SWP

Each SWP module, illustrated in Figure 6(c), is a predictor that makes a soft or hard write prediction for every dynamic store instruction executed in one of the CPU's cores. (There are multiple SWP modules in the CPU–one per core located alongside L1-cache). The predictor we use is adapted from the gskew branch predictor [51]. To mitigate aliasing, three separate tables are employed in each SWP module, with each

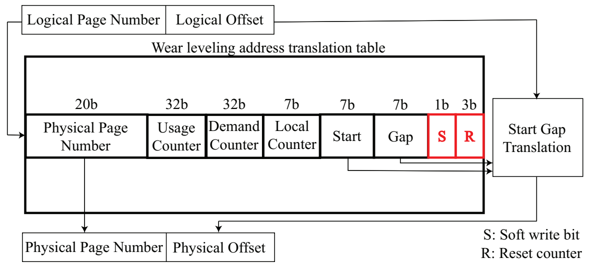

Figure 7: Wear leveling table with added fields (in red) for SRTP refresh and reset operations.

table containing a column of 3-bit saturating counters. Store PCs are used to lookup the predictor. Each lookup indexes into all three tables, but a different hash function is applied to the store PC to compute the per-table indices. (So, collisions may occur in one table but are unlikely to occur in two or three tables).

As the RTD computes soft write decisions, it trains the SWP by looking up the 3 counters associated with the tracked store PC (label ⑤ in Figure 6(c)) and incrementing each one of them. The RTD output can be sent to SWP using the on chip interconnect between LLC and L1-cache. All the counters are initialized to 0, corresponding to a hard write prediction, but flip to a soft write prediction once the count reaches 2. Thus, two soft write outcomes for the same store PC are needed from the RTD to initiate soft writes for the corresponding static store instruction.

Simultaneous with training, each CPU core looks up the SWP whenever a store instruction executes using the store's PC (label ⑥ in Figure 6(c)). Each of the 3 counters in the predictor associated with the store PC is then compared to the threshold value (2) to determine whether the counter reflects a soft or hard write prediction. A majority vote across the 3 counters determines the final soft write prediction for the store. (By using all 3 counters, destructive aliasing in one table will not negatively affect the final prediction).

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 in Figure 6(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 field 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 to main memory with the appropriate write strength (either soft or hard).

4.3 Refresh, Resets, and Wear Leveling

In addition to soft write prediction, SRTP must also track the memory pages receiving soft writes for refresh and reset purposes. As our results will show, SRTP performs many more soft writes than RRM, which increases the number of pages that must be tracked. We find SRTP can have 100s of thousands of softly written pages (compared to just 4K or 32K pages for RRM-base and RRM-aggr). Due to its size, we implement the tracking data structure in main memory.

The tracking data structure contains two fields. First, there is a "soft write bit" for each page that indicates whether the page has been softly written. This bit is similar to RRM's hot bit in that it identifies the pages requiring refresh operations. When a page receives a soft LLC writeback, 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, SRTP refreshes the entire page–it does not further distinguish softly written memory blocks within a page as RRM does to minimize the tracking overhead.

SRTP's tracking data structure also contains a 3-bit "reset counter" per page. This counter is similar to RRM's decay_counter, and is used to determine when softly written pages stop receiving soft writes. It is incremented when the page is refreshed, but is reset to zero after every 100 soft writes to the page. If the page stops receiving soft writes, then the count will continue to rise, signifying that refreshes are being incurred without any soft write benefit. When the count reaches a threshold (7), SRTP performs a reset hard write: it issues a hard write to every memory block in the page, clears the soft write bit, and stops refreshing the page.

To implement the soft write bits and reset counters, a separate standalone table could be created in main memory. However, we propose to implement SRTP's tracking data structure as part of an integrated wear leveling scheme. Wear leveling is still a problem in soft writable memory systems (soft writes do not affect how the writes are distributed across memory). Yet, to our knowledge, no prior soft write technique has considered integration with wear leveling.

The simplest wear leveling techniques, such as Start-Gap, use algebraic expressions to remap memory locations [23, 44, 64, 68, 78, 90]. This approach is good at wear leveling in smaller regions, but because the line movements are not targeted, there is a chance 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 [15, 36, 70, 95, 104]. These techniques are able to 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 [45]. Ouroboros divides physical memory into regions and uses table-based wear leveling across regions. Within a region, Ouroboros uses Start-Gap to provide wear leveling locally. Figure 7 shows the table entry format used in Ouroboros.

Ouroboros uses Start-Gap to shift lines in a circular fashion within a region–*i.e.*, page–after a certain number of local writes (100), tracked by the "local counter." After a fixed number of global writes (100 million), Ouroboros migrates a small number of logical pages (10) that had the highest demand to physical pages with the lowest usage. The "demand counter" keeps track of the writes to a logical page since its last migration, and the "usage counter" keeps track of the overall usage of the physical page.

We propose to integrate SRTP's refresh and reset information into the wear leveling table, which already tracks write information on a per-page basis. Figure 7 shows the addition to each Ouroboros table entry in red. The extension of wear leveling to handle refresh and resets only incurs a 3.8%

CPU	
Number of cores	4
Core	x86, out-of-order, 2GHz
L1 Caches	Private, 32 KB I/D, 4-way, 3-cycle hits
L2 Caches	Private, 256 KB, 8-way, 7-cycle hits
L3 Cache	Shared, 16 MB, 16-way, 27-cycle hits
cache block size	64 B
Main Memory	
Memory	667MHz, 8 KB page size, 8 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	2×10^6 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 time	10 seconds
Wear Leveling	Local threshold 100 writes, global threshold 10^8 writes, 10 migrations/global epoch, 128 spare frames

Table 1: Simulation parameters.

increase in the table size.

5. EXPERIMENTAL METHODOLOGY

Architectural simulators are typically used to study events that exhibit very short time scales. In our work, we are interested in studying ReRAM retention events that can span 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 reuse times that drive soft write decisions. We use the Zsim multicore simulator [67] for our evaluation. Currently, Zsim is one of the fastest simulators available, and can be used to study the time scales demanded by our research. Zsim also provides detailed architecture models that enable cycle-accurate simulation. In particular, we configured Zsim to model a quad-core CPU with out-of-order cores and a 3-level cache hierarchy. The top portion of Table 1 reports the CPU parameters that we used in our experiments.

The main modification we made to Zsim is to add support for our SRTP technique. Specifically, we added the RTD and SWP modules described in Section 4. In our experiments, we model a 512-entry RTD and a 1024-entry SWP, with each SWP module containing 3072 total entries across its 3 tables. We also added to Zsim the RRM technique discussed in Section 3.1–and modeled two versions of RRM–so that we can compare SRTP against this prior state-of-the-art technique. (The RRM and SRTP parameters are given in Tables 3 and 4, and will be discussed below). Lastly, to enable a limit study, we implemented the Oracle policy from Section 3.2. This requires allocating a separate region of memory to shadow main memory, providing space for the per-memory block timestamps that the Oracle requires.

Something lacking in Zsim are models of non-volatile memory. While there exist NVM simulators, none of them are fast enough to be integrated into Zsim; doing so would completely sacrifice the ability to study ReRAM retention events. Fortunately, Zsim exposes many detailed memory

Benchmark	WPKI	Inst (T)	Benchmark	WPKI	Inst (T)
lbm	17.8	7.1	xalancbmk	0.76	5.1
fotonik3d	11.4	6.1	deepsjeng	0.43	7.4
omnetpp	5.5	4.2	nab	0.36	8
roms	3.84	8	parest	0.3	8
mcf	3.49	3.9	namd	0.21	8
cactuBSSN	2.25	6.1	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

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

timing parameters that can be configured. We set these parameters to match models available in NVMain [61], a recent non-volatile memory simulator, to accurately reflect the timing characteristics of a ReRAM-based main memory system.

Many different latency, energy, and endurance numbers have been reported for ReRAM in the literature. In our evaluation, we use a base latency (without contention) of 210ns and 420ns for reads and writes, respectively [77]. Read energies in the range 1-2.4 pJ/bit [57, 77] and write energies in the range 4.8-53 pJ/bit [6, 38, 77, 100] have been reported. We use 2 pJ/bit and 30 pJ/bit for the read and write energies, respectively. Lastly, endurance values in between 10^5 and 10^7 have been provided [7, 10, 16, 20, 96].We use an endurance of 2×10^6 hard writes per bitcell. The bottom portion of Table 1 reports all of the timing, energy, and endurance parameters we used in the memory system.

As discussed earlier, SRTP 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 is 10x while the SWA for energy (Equation 4) is only 6.4x. Hence, by optimizing for endurance, our technique will perform too many soft writes from an energy standpoint. Nevertheless, we will still provide energy benefits, so Section 6 reports both endurance and energy results even though we explicitly optimize for endurance.

To drive our simulations, we use the SPEC CPU2017 benchmarks [72]. The 20 benchmarks employed in our study represent the complete SPEC CPU2017 suite, except for `gcc` which failed to run on Zsim, and `povray` and `exchange2` which had barely any writebacks. Table 2 lists the benchmarks in order of their memory intensity. (The benchmarks are sorted on column two of Table 2 which reports the number of writebacks per 1000 instructions, or "WPKI"). We compile the benchmarks using the Gcc compiler with the highest level of optimization. In every case, we run the benchmark to completion. On average, the benchmarks were simulated for 6.5 *trillion* instructions, and exhibited a wall clock time of 1,584 seconds (about 26 minutes). 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 extrapolation experiments.

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

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

Table 3: RRM parameters and overhead.

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

Hardware Cost. Before presenting results, we compare the hardware costs for RRM and SRTP. Table 3 shows the configuration parameters for two versions of RRM that we evaluate. "RRM-base" has 4,096 entries (256 sets $\times$ 16 ways), requiring a 98 KB table. This is identical to the hardware from the original RRM paper [98]. "RRM-aggr" is a more aggressive version of the scheme with 8x more entries–32,768 entries–and 8x the table size–784 KB. SRTP, on the other hand, adds two hardware structures, the RTD and SWP modules, and tracking in the cache tags. These are the structures residing on the CPU die, and they amount to 69.8 KB. SRTP also requires 512 KB to track refreshes and resets. This table is large because SRTP is very successful at capitalizing on soft write opportunities. However, it can be implemented off-chip in the non-volatile memory because it is accessed infrequently. Moreover, if we integrate SRTP with Ouroboros, the refresh/reset bits can be folded into the wear leveling tables with minimal overhead (see Section 4.3).

SRTP modules area and energy. An individual SWP table is same as the local predictor in tournament branch predictor [34]. We use McPat [42] to get the area and power of this local predictor in Xeon cores at 22nm technology node. By extending these numbers for three tables we get 0.028mm^2 and 0.05W as area and peak power, respectively, for each SWP module. Similar calculations are undertaken for RTD using Cacti [43]. The RTD module needs 0.43mm^2 area at 22nm. For SPEC CPU2017 benchmarks on an average RTD will consume 60μW of dynamic power and 20mW of leakage. The dynamic power is low because sampling cuts down a lot of activity in RTD by accessing it only for $\frac{1}{32}$ of the writebacks from LLC. Since, the energy consumption of SRTP is tiny we can easily ignore it in the energy calculations in Section 6.

6. EXPERIMENTAL EVALUATION

This section presents our experimental evaluation of SRTP. We begin by quantifying SRTP's improvement in endurance, which is the objective function that our experiments target. Figure 8 presents these endurance results. In Figure 8, we plot the effective SWA_{end} achieved by SRTP and compare it against the Oracle. The graph uses the exact same format as Figure 4, so the two can be directly compared. (In fact, the Oracle bars in Figures 4 and 8 are identical). As already mentioned in Section 3.2, the Oracle policy achieves an effective SWA_{end} of 7.6x. Figure 8 shows SRTP's effective SWA_{end} is within 18.5% of the oracle at 6.2x. This is a significant

RTD		SWP	
Sets	32	SWP count	4
PC Associativity (20 bits)	16	Saturating counter entry size	3 bits
Addresses/PC (48 bits)	2		
Timestamps/PC (32 bits)	2	#Saturating counter entries	1024
SRAM Overhead	**11.7KB**	SRAM Overhead	**1.1 KB**
Caches		**Wear Leveling Table**	
Soft Write Bit (S)	1/entry	Soft write bit	1/entry
Last write PC (20 bits)	1/32 entries	Reset counter (3 bits)	1/entry
SRAM Overhead	**57 KB**	NVM Overhead	**512 KB**
Total Overhead	**69.8 KB** SRAM and **512 KB** NVM		

Table 4: SRTP parameters and overheads.

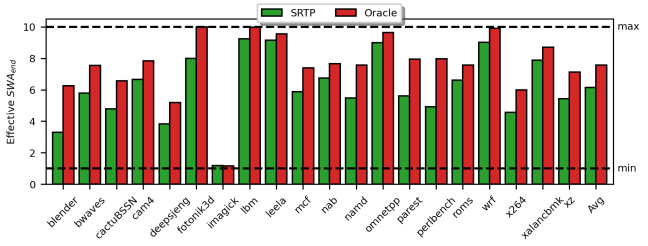

Figure 8: Effective SWA_{end} for SRTP and the Oracle.

improvement over RRM-base and RRM-aggr from Figure 4, whose effective SWA_{end} are just 1.5x and 2.4x, respectively.

To provide further insights, Figure 9 breaks down the different types of writes performed by RRM (base and aggressive), SRTP, and the Oracle. In addition to the soft write, refresh, and hard write categories, Figure 9 also breaks down reset hard writes into "eviction" and "decay" categories. The eviction category includes resets that occur due to a shortage of tracking entries. The decay category includes resets that occur when a technique predicts it is no longer profitable to continue issuing soft writes to certain memory locations. All bars are normalized to the total number of writebacks induced by the application. (Because refreshes and resets are in addition to the applications' normal writebacks, the breakdowns add up to more than 100%).

As Figure 9 shows, SRTP does two things very well. First, it is able to identify and convert a large fraction of writes into soft writes. On average, 92% of the application-level writes for SRTP are soft writes, whereas the percentage of soft writes for RRM-base and RRM-aggr is only 30% and 51%, respectively. SRTP is also very close to the Oracle, which performs 95% of the application-level writes as soft writes. This is mainly why SRTP's endurance results in Figure 8 are superior to RRM and nearly match that of the Oracle. Second, SRTP also has a very small number of reset hard writes, just 0.6%, whereas the RRM policies have many more reset hard writes. In particular, RRM-base and RRM-aggr incur 17% and 5.8% resets, respectively, primarily due to evictions. (By definition, the Oracle has zero resets since it omnisciently knows when to perform hard writes).

SRTP uncovers practically all soft write opportunities because it accurately predicts store reuse time, and applies the soft write criterion–Inequality 1–to precisely determine the profitability of soft writes in a fine-grain manner. This mimics the behavior of the Oracle. In contrast, RRM's soft write

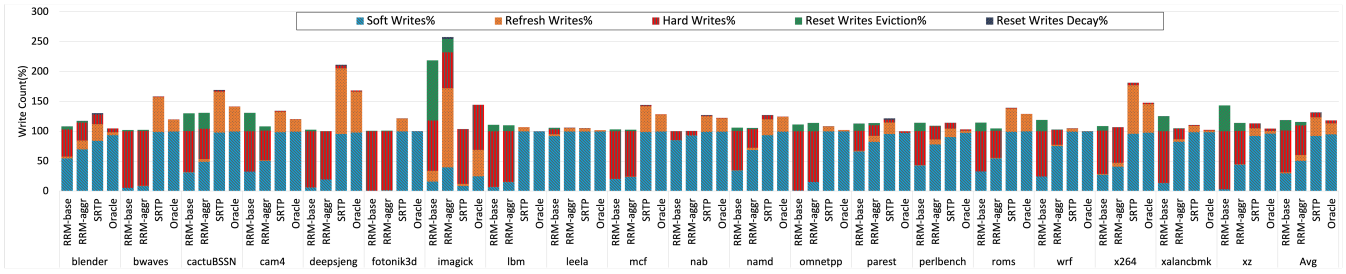

Figure 9: Breakdown of different types of writes normalized against the writes without soft write techniques.

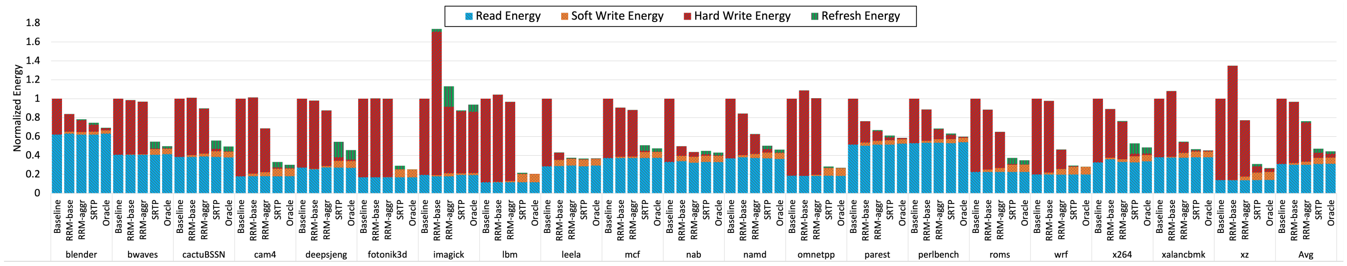

Figure 10: Breakdown of memory system energy normalized against main memory energy with only hard writes.

decisions based on coarse-grained page access counts are inaccurate, missing numerous 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 exhibit the store locality that warrants employing soft writes.

In addition to higher accuracy, SRTP also exhibits lower predictor overhead. The RTD consumes only 11.7 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 RTD is able to track almost all last-touch store PCs without incurring many evictions. In contrast, RRM (and especially RRM-base) exhibits numerous evictions, as is evident from the reset eviction components in Figure 9. Overall, SRTP's CPU-side prediction mechanism is much lighter weight compared to RRM's memory-side predictor. There are simply many more memory locations written to than there are store PCs writing them.

And lastly, SRTP'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, SRTP only needs to train on the last-touch store PCs, which can happen rapidly because there are so few of them. This allows SRTP to make correct soft write decisions much sooner.

Although our experiments use Inequality 1 to explicitly optimize for endurance (instead of Inequality 5 to optimize for energy), they still show significant energy improvements. Figure 10 presents our energy results. In Figure 10, we plot the total energy incurred in the memory system normalized to the energy consumed by the baseline system that always performs hard writes. The figure also breaks down the total energy into reads and different types of writes. (Reset hard writes are folded into the hard write category).

Even though there are 4.5x fewer writes than reads, writes still consume 76% of the memory system energy, on average,

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.

as the breakdown of the "Baseline" bars in Figure 10 shows. For the most write intensive benchmark, lbm, writes account for 92% of the energy. Unfortunately, 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, SRTP reduces memory system energy by 2.4x thanks to a 4.5x reduction in write energy. This puts SRTP 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 11 plots the endurance results for these workloads in the exact same format as Figures 4 and 8.

Figure 11 shows SRTP does even better on the mixed workloads. On average, SRTP achieves an effective SWA_{end} of 6.8x, up from 6.2x in Figure 8. And, SRTP comes within 19.1% of the Oracle (which has an effective SWA_{end} of 8.4x),

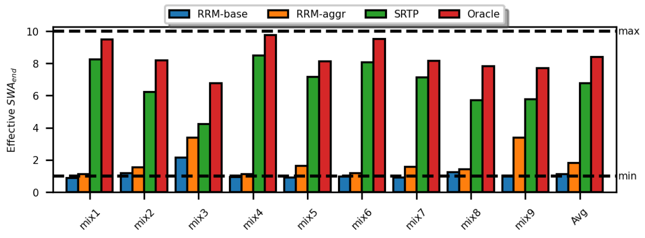

Figure 11: Effective SWA_{end} for mixed workloads.

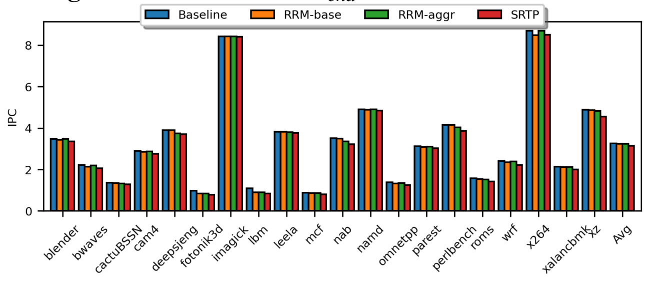

Figure 12: Performance impact of soft write techniques.

down from 18.5% in Figure 8. Also, SRTP is 6x and 3.8x better than RRM-base and RRM-aggr, respectively, up from 4.1x and 2.6x across Figures 4 and 8. SRTP'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, SRTP performs even better for multiprogrammed workloads maintaining its lead over RRM in all cases. Although we omit the detailed results to save space, energy improvements are also better for the mixed workloads, with SRTP reducing total main memory energy by 3x.

6.2 Performance Results

Soft write techniques improve energy and endurance, but they also add extra refreshes and resets. Although these additional memory operations are off the critical path of CPU accesses, they can still affect performance by increasing contention in the memory system. Figure 12 reports the IPC for the different techniques we evaluate. As Figure 12 shows, all of the soft write techniques incur a very small performance penalty, indicating that contention effects are minimal. In the case of SRTP, there is a 4.3% performance degradation averaged across all 20 SPEC benchmarks. SRTP's performance hit is the largest since it is also the most aggressive in issuing soft writes (and hence, incurs the most refreshes). But, SRTP's performance impact is minimal relative to the energy and endurance gains it provides.

6.3 Wear Leveling Results

Here we analyze the efficacy of SRTP when integrated with the Ouroboros wear leveling technique from Section 4.3 and actual lifetime achieved. Since Start-Gap has been demonstrated to be highly effective in regions much larger than our page size, 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 a benchmark at every global migration epoch of Ouroboros (100 million writes). These profiles are run repeatedly on a program that performs Ouroboros's page migration

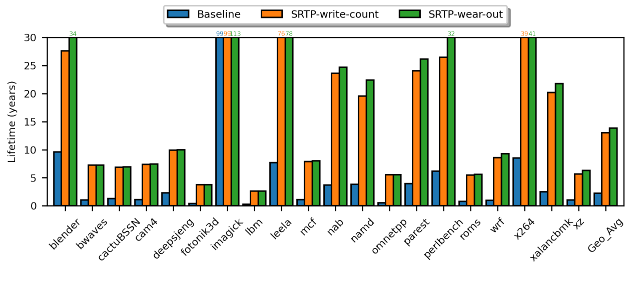

Figure 13: Extrapolated lifetime

until all the spare frames (128) are worn out. The number of runs of the benchmark from extrapolation and the runtime for a single simulated run gives us the lifetime of the memory system if the program was run repeatedly until failure.

When integrating with wear leveling techniques, we propose to use wear out as a guiding metric instead of write count given that soft and hard writes cause uneven wear out. Figure 9 shows that for SRTP, hard writes account for only 6.3% of total writes on average, but they are responsible for 40% of wear out. The traditional policy of just considering write count will negatively impact the lifetime a wear leveling technique can achieve. To make Ouroboros consider wear out, demand and usage counters are incremented by 1 for soft writes and by 10 for hard writes.

Figure 13 compares SRTP's lifetime against the baseline system with just hard writes. SRTP lifetime accounts for wear leveling via extrapolation while the baseline assumes perfect wear leveling. (So, the gains for SRTP over baseline are conservative). For SRTP, we report the lifetime when using the write count metric versus the wear out metric to guide wear leveling, labeled "SRTP-write-count" and "SRTP-wear-out," respectively. On average, we see 6.1% improvement in lifetime–from 13 years to 13.8 years–due to using the wear out metric. Overall, SRTP improves average lifetime of ReRAM from 2.2 years (for the baseline) to 13.8 years, a gain of 6.3x. Overall, the 6.2x effective SWA_{end} translates to 6.3x improvement in memory lifetime.

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 [41, 60, 63], NVM as a cache for SSDs [32], persistent programming frameworks [47], and architectural techniques [97, 98]. Of these, RRM is the best-performing technique that (like our work) targets systems with CPUs and non-volatile main memory [98]. We adapted RRM to control soft writes for single-level cell ReRAM main memory to optimize energy and endurance. Our results show SRTP improves upon RRM significantly.

Mellow Writes [31, 96] 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, SRTP 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, SRTP and Mellow Writes are orthogonal. SRTP 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 non-volatile memory. Another direction of work tries to reduce the *number of writes*. This includes techniques that only write the modified bits [5, 8, 12, 19, 89], encode the data [14, 28, 49, 55, 59, 80, 87, 101], compress the data [58, 86], or utilize asymmetries in set and reset operations [8, 71, 89, 92]. All of these techniques are orthogonal to our work and can be applied on top of soft writes.

Another write reduction technique is LLC management policies to reduce the number of writebacks [3, 62, 82, 94, 103]. These policies can only target writebacks with short reuse times, leaving a lot of soft write opportunities. Since SRTP is agnostic to LLC management, it can adjust to these techniques as long as the RTD gets the needed information.

8. CONCLUSION

This paper exploits the retention versus energy and endurance tradeoff that exists in ReRAM. Our work identifies the relationship between store reuse time, retention time, and soft write advantage that governs the profitability of using soft writes for each LLC writeback. We develop an architectural technique, called SRTP, that learns the dominant store reuse 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 conduct a simulation-based evaluation of SRTP, and show that it improves endurance by 2.6x to 4.1x compared to a state-of-the-art soft write technique, RRM. SRTP also comes within 18.5% of the Oracle's endurance. Finally, we integrate SRTP with an existing wear leveling technique and demonstrate that SRTP can improve actual lifetime by 5.0x.

REFERENCES

[1] N. Agarawal and T. F. Wenisch, "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*, 2017.

[2] M. Arjomand, M. T. Kandemir, A. Sivasubramaniam, and C. R. Das, "Boosting access parallelism to pcm-based main memory," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2016, pp. 695–706.

[3] M. Bakhshalipour, A. Faraji, S. A. V. Ghahani, F. Samandi, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Reducing writebacks through in-cache displacement," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 24, no. 2, pp. 1–21, 2019.

[4] J. Chang and G. S. Sohi, "Cooperative Cache Partitioning for Chip Multiprocessors," in *Proceedings of the International Conference on Supercomputing*, Seattle, WA, June 2007.

[5] J. Chen, R. C. Chiang, H. H. Huang, and G. Venkataramani, "Energy-aware writes to non-volatile main memory," *ACM SIGOPS Operating Systems Review*, vol. 45, no. 3, pp. 48–52, 2012.

[6] Y.-C. Chen, "Selector-less resistive random access memory (rram) with intrinsic nonlinearity for crossbar array applications," Ph.D. dissertation, 2019.

[7] Z. Chen, H. Wu, B. Gao, D. Wu, N. Deng, H. Qian, Z. Lu, B. Haukness, M. Kellam, and G. Bronner, "Performance improvements by sl-current limiter and novel programming methods on 16mb rram chip," in *2017 IEEE International Memory Workshop (IMW)*. IEEE, 2017, pp. 1–4.

[8] S. Cho and H. Lee, "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*, 2009, pp. 347–357.

[9] S. Clima, Y. Chen, A. Fantini, L. Goux, R. Degraeve, B. Govoreanu, G. Pourtois, and M. Jurczak, "Intrinsic tailing of resistive states distributions in amorphous hfo x and tao x based resistive random access memories," *IEEE Electron Device Letters*, vol. 36, no. 8, pp. 769–771, 2015.

[10] Crossbar, "Crossbar ReRAM Overview." 2020. [Online]. Available: https://www.crossbar-inc.com/technology/reram-overview/

[11] C. Ding and Y. Zhong, "Predicting whole-program locality through reuse distance analysis," in *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, 2003, pp. 245–257.

[12] W. Dong, X. Li, Y. Li, M. Qiu, L. Dou, L. Ju, and Z. Jia, "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, 2015, pp. 290–295.

[13] S. R. Dulloor, A. Roy, Z. Zhao, N. Sundaram, N. Satish, R. Sankaran, J. Jackson, and K. Schwan, "Data Tiering in Heterogeneous Memory Systems," in *Proceedings of the 2016 EuroSys Conference*, 2016.

[14] D. Feng, J. Xu, Y. Hua, W. Tong, J. Liu, C. Li, and Y. Chen, "A low-overhead encoding scheme to extend the lifetime of nonvolatile memories," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2516–2529, 2019.

[15] A. P. Ferreira, M. Zhou, S. Bock, B. Childers, R. Melhem, and D. Mossé, "Increasing pcm main memory lifetime," in *2010 Design, Automation Test in Europe Conference Exhibition (DATE 2010)*, 2010, pp. 914–919.

[16] A. Grossi, E. Vianello, M. M. Sabry, M. Barlas, L. Grenouillet, J. Coignus, E. Beigne, T. Wu, B. Q. Le, M. K. Wootters *et al.*, "Resistive ram endurance: Array-level characterization and correction techniques targeting deep learning applications," *IEEE Transactions on Electron Devices*, vol. 66, no. 3, pp. 1281–1288, 2019.

[17] T. J. Ham, B. K. Chelepalli, N. Xue, and B. C. Lee, "Disintegrated control for energy-efficient and heterogeneous memory systems," in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2013, pp. 424–435.

[18] H. Hassan, G. Pekhimenko, N. Vijaykumar, V. Seshadri, D. Lee, O. Ergin, and O. Mutlu, "Chargecache: Reducing dram latency by exploiting row access locality," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2016, pp. 581–593.

[19] A. Hay, K. Strauss, T. Sherwood, G. H. Loh, and D. Burger, "Preventing pcm banks from seizing too much power," in *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2011, pp. 186–195.

[20] Y. Hayakawa, A. Himeno, R. Yasuhara, W. Boullart, E. Vecchio, T. Vandeweyer, T. Witters, D. Crotti, M. Jurczak, S. Fujii *et al.*, "Highly reliable tao x reram with centralized filament for 28-nm embedded application," in *2015 Symposium on VLSI Technology (VLSI Technology)*. IEEE, 2015, pp. T14–T15.

[21] T. Heo, Y. Wang, W. Cui, J. Huh, and L. Zhang, "Adaptive page migration policy with huge pages in tiered memory systems," *IEEE Transactions on Computers*, vol. 71, no. 1, pp. 53–68, 2020.

[22] T. Hirofuchi and R. Takano, "RAMinate: Hypervisor-based Virtualization for Hybrid Main Memory Systems," in *Proceedings of the ACM Symposium on Cloud Computing*, Santa Clara, CA, October 2016.

[23] J. Huang, Y. Hua, P. Zuo, W. Zhou, and F. Huang, "An efficient wear-level architecture using self-adaptive wear leveling," in *49th International Conference on Parallel Processing - ICPP*, ser. ICPP '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3404397.3404405

[24] D. Ielmini, F. Nardi, C. Cagli, and A. L. Lacaita, "Trade-off Between

Data Retention and Reset in NIO RRAMs," in *Proceedings of the IEEE International Reliability Physics Symposium*, May 2010.

[25] Intel, "Intel Optane Technology," 2017. [Online]. Available: http://www.intel.com/content/www/us/en/architecture-and-technology/intel-optane-technology.html

[26] E. Ipek, J. Condit, E. B. Nightingale, D. Burger, and T. Moscibroda, "Dynamically replicated memory: building reliable systems from nanoscale resistive memories," *ACM Sigplan Notices*, vol. 45, no. 3, pp. 3–14, 2010.

[27] 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.

[28] A. N. Jacobvitz, R. Calderbank, and D. J. Sorin, "Coset coding to extend the lifetime of memory," in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2013, pp. 222–233.

[29] M. Jagasivamani, "Resistive ram based main-memory systems: Understanding the opportunities, limitations, and tradeoffs," Ph.D. dissertation, University of Maryland, College Park, 2020. [Online]. Available: https://drum.lib.umd.edu/handle/1903/26228

[30] A. Jaleel, K. B. Theobald, S. C. S. Jr., and J. Emer, "High Performance Cache Replacement Using Re-Reference Interval Prediction (RRIP)," in *Proceedings of the International Symposium on Computer Architecture*, Saint Malo, France, June 2010.

[31] M. R. Jokar, L. Zhang, and F. T. Chong, "Cooperative nv-numa: prolonging non-volatile memory lifetime through bandwidth sharing," in *Proceedings of the International Symposium on Memory Systems*, 2018, pp. 67–78.

[32] D. Kang, S. Baek, J. Choi, D. Lee, S. H. Noh, and O. Mutlu, "Amnesic cache management for non-volatile memory," in *2015 31st Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 2015, pp. 1–13.

[33] G. Keramidas, P. Petoumenos, and S. Kaxiras, "Cache replacement based on reuse-distance prediction," in *2007 25th International Conference on Computer Design*. IEEE, 2007, pp. 245–250.

[34] R. Kessler, E. McLellan, and D. Webb, "The alpha 21264 microprocessor architecture," in *Proceedings International Conference on Computer Design. VLSI in Computers and Processors (Cat. No.98CB36273)*, 1998, pp. 90–95.

[35] S. Kim, D. Chandra, and Y. Solihin, "Fair Cache Sharing and Partitioning in a Chip Multiprocessor Architecture," in *Proceedings of the International Symposium on High Performance Computer Architecture*, 2002.

[36] S. Kim, H. Jung, W. Shin, H. Lee, and H.-J. Lee, "Had-twl: Hot address detection-based wear leveling for phase-change memory systems with low latency," *IEEE Computer Architecture Letters*, vol. 18, no. 2, pp. 107–110, 2019.

[37] C. Lammie, M. R. Azghadi, and D. Ielmini, "Empirical Metal-Oxide RRAM Device Endurance and Retention Model for Deep Learning Simulations," *Semiconductor Science and Technology*, vol. 36, 2021.

[38] A. Lee, C.-P. Lo, C.-C. Lin, W.-H. Chen, K.-H. Hsu, Z. Wang, F. Su, Z. Yuan, Q. Wei, Y.-C. King *et al.*, "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*, vol. 52, no. 8, pp. 2194–2207, 2017.

[39] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable dram alternative," in *Proceedings of the 36th annual international symposium on Computer architecture*, 2009, pp. 2–13.

[40] B. C. Lee, P. Zhou, J. Yang, Y. Zhang, B. Zhao, E. Ipek, O. Mutlu, and D. Burger, "Phase-change technology and the future of main memory," *IEEE micro*, vol. 30, no. 1, pp. 143–143, 2010.

[41] Q. Li, L. Jiang, Y. Zhang, Y. He, and C. J. Xue, "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*, 2013, pp. 101–110.

[42] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "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*, 2009, pp. 469–480.

[43] S. Li, K. Chen, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "Cacti-p: Architecture-level modeling for sram-based structures with advanced leakage reduction techniques," in *ICCAD: International Conference on Computer-Aided Design*, pp. 694–701.

[44] D. Liu, T. Wang, Y. Wang, Z. Shao, Q. Zhuge, and E. Sha, "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)*, 2013, pp. 279–284.

[45] Q. Liu and P. Varman, "Ouroboros wear leveling for nvram using hierarchical block migration," *ACM Transactions on Storage (TOS)*, vol. 13, no. 4, pp. 1–31, 2017.

[46] Q. Liu and P. Varman, "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, May 2017.

[47] R.-S. Liu, D.-Y. Shen, C.-L. Yang, S.-C. Yu, and C.-Y. M. Wang, "Nvm duet: Unified working memory and persistent store architecture," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 1, pp. 455–470, 2014.

[48] W. Liu and D. Yeung, "Enhancing ltp-driven cache management using reuse distance information," *J. Instr. Level Parallelism*, vol. 11, 2009.

[49] H. Luo, L. Shi, Q. Li, C. J. Xue, and E. H.-M. Sha, "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, 2018, pp. 554–559.

[50] K. Ma, X. Li, K. Swaminathan, Y. Zheng, S. Li, Y. Liu, Y. Xie, J. J. Sampson, and V. Narayanan, "Nonvolatile processor architectures: Efficient, reliable progress with unstable power," *IEEE Micro*, vol. 36, no. 3, pp. 72–83, 2016.

[51] P. Michaud, A. Seznec, and R. Uhlig, "Trading Conflict and Capacity Aliasing in Conditional Branch Predictors," in *Proceedings of the 24th International Symposium on Computer Architecture*, 1997.

[52] S. Mirbagher-Ajorpaz, E. Garza, G. Pokam, and D. A. Jiménez, "Chirp: Control-flow history reuse prediction," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 131–145.

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

[54] P. J. Nair, D.-H. Kim, and M. K. Qureshi, "Archshield: Architectural framework for assisting dram scaling by tolerating high error rates," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, pp. 72–83, 2013.

[55] K. Namba and F. Lombardi, "A coding scheme for write time improvement of phase change memory (pcm) systems," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 2, no. 4, pp. 291–296, 2016.

[56] F. Nardi, D. Ielmini, C. Cagli, S. Spiga, M. Fanciulli, L. Goux, and D. Wouters, "Control of filament size and reduction of reset current below 10 μa in nio resistance switching memories," *Solid-State Electronics*, vol. 58, no. 1, pp. 42–47, 2011.

[57] D. Niu, C. Xu, N. Muralimanohar, N. P. Jouppi, and Y. Xie, "Design of cross-point metal-oxide reram emphasizing reliability and cost," in *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2013, pp. 17–23.

[58] 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.

[59] P. M. Palangappa and K. Mohanram, "Rapid: read acceleration for improved performance and endurance in mlc/tlc nvms," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–7.

[60] C. Pan, M. Xie, J. Hu, Y. Chen, and C. Yang, "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*, 2014, pp. 1–10.

[61] M. Poremba, T. Zhang, and Y. Xie, "Nvmain 2.0: A user-friendly memory simulator to model (non-) volatile memory systems," *IEEE Computer Architecture Letters*, vol. 14, no. 2, pp. 140–143, 2015.

[62] B. Pourshirazi, M. V. Beigi, Z. Zhu, and G. Memik, "Wall: A writeback-aware llc management for pcm-based main memory systems," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 449–454.

[63] K. Qiu, Q. Li, and C. J. Xue, "Write mode aware loop tiling for high performance low power volatile pcm," in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2014, pp. 1–6.

[64] M. K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali, "Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling," in *2009 42nd Annual IEEE/ACM international symposium on microarchitecture (MICRO)*. IEEE, 2009, pp. 14–23.

[65] M. K. Qureshi and Y. N. Patt, "Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches," in *Proceedings of the International Symposium on Microarchitecture*, 2006.

[66] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable High Performance Main Memory System Using Phase-Change Memory Technology," in *Proceedings of the International Symposium on Computer ARchitecture*, Austin, TX, 2009.

[67] D. Sanchez and C. Kozyrakis, "Zsim: Fast and accurate microarchitectural simulation of thousand-core systems," *ACM SIGARCH Computer architecture news*, vol. 41, no. 3, pp. 475–486, 2013.

[68] N. H. Seong, D. H. Woo, and H.-H. S. Lee, "Security refresh: Prevent malicious wear-out and increase durability for phase-change memory with dynamically randomized address mapping," *ACM SIGARCH computer architecture news*, vol. 38, no. 3, pp. 383–394, 2010.

[69] V. Seshadri, O. Mutlu, M. A. Kozuch, and T. C. Mowry, "The Evicted-Address Filter: A Unified Mechanism to Address Both Cache Pollution and Thrashing," in *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*, Minneapolis, MN, September 2012.

[70] A. Seznec, "A phase change memory as a secure main memory," *IEEE Computer Architecture Letters*, vol. 9, no. 1, pp. 5–8, 2010.

[71] S. Song, A. Das, O. Mutlu, and N. Kandasamy, "Improving phase change memory performance with data content aware access," in *Proceedings of the 2020 ACM SIGPLAN International Symposium on Memory Management*, 2020, pp. 30–47.

[72] "SPEC CPU2017 Benchmarks. ," 2017. [Online]. Available: https://www.spec.org/cpu2017/

[73] D. Strukov, "The area and latency tradeoffs of binary bit-parallel bch decoders for prospective nanoelectronic memories," in *2006 Fortieth Asilomar Conference on Signals, Systems and Computers*. IEEE, 2006, pp. 1183–1187.

[74] D. Ustiugov, M. Suterhland, A. Daglis, E. Bugnion, D. Pnevmatikatos, J. Picorel, and B. Falsafi, "Design Guidelines for High-Performance SCM Hierarchies," in *Proceedings of the 4th International Symposium on Memory Systems*, National Harbor, D.C., 2018.

[75] M. Valad Beigi, B. Pourshirazi, G. Memik, and Z. Zhu, "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*, 2020, pp. 353–354.

[76] I. Valov, R. Waser, J. R. Jameson, and M. N. Kozicki, "Electrochemical metallization memories—fundamentals, applications, prospects," *Nanotechnology*, vol. 22, no. 25, p. 254003, 2011.

[77] C. Walden, D. Singh, M. Jagasivamani, S. Li, L. Kang, M. Asnaashari, S. Dubois, B. Jacob, and D. Yeung, "Monolithically integrating non-volatile main memory over the last-level cache," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 18, no. 4, pp. 1–26, 2021.

[78] G. Wang, F. Peng, L. Ju, L. Zhang, and Z. Jia, "Double circulation wear leveling for pcm-based embedded systems," in *Advanced Computer Architecture*. Springer, 2014, pp. 190–200.

[79] H. Wang, J. Zhang, S. Shridhar, G. Park, M. Jung, and N. S. Kim, "Duang: Fast and lightweight page migration in asymmetric memory systems," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2016, pp. 481–493.

[80] J. Wang, X. Dong, G. Sun, D. Niu, and Y. Xie, "Energy-efficient multi-level cell phase-change memory system with data encoding," in *2011 IEEE 29th International Conference on Computer Design (ICCD)*. IEEE, 2011, pp. 175–182.

[81] Y. Wang, A. Tavakkol, L. Orosa, S. Ghose, N. M. Ghiasi, M. Patel, J. S. Kim, H. Hassan, M. Sadrosadati, and O. Mutlu, "Reducing dram latency via charge-level-aware look-ahead partial restoration," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 298–311.

[82] Z. Wang, S. Shan, T. Cao, J. Gu, Y. Xu, S. Mu, Y. Xie, and D. A. Jiménez, "Wade: Writeback-aware dynamic cache management for nvm-based main memory system," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 10, no. 4, pp. 1–21, 2013.

[83] C.-J. Wu, A. Jaleel, W. Hasenplaugh, M. Martonosi, S. C. S. Jr., and J. Emer, "SHiP: Signature-based Hit Predictor for High Performance Caching," in *Proceedings of the International Symposium on Microarcitecture*, Porto Alegre, Brazil, December 2011.

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

[85] C. Xu, D. Niu, N. Muralimanohar, N. P. Jouppi, and Y. Xie, "Understanding the trade-offs in multi-level cell reram memory design," in *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2013, pp. 1–6.

[86] J. Xu, D. Feng, Y. Hua, W. Tong, J. Liu, and C. Li, "Extending the lifetime of nvms with compression," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 1604–1609.

[87] J. Xu, D. Feng, Y. Hua, W. Tong, J. Liu, C. Li, and W. Zhou, "Improving performance of tlc rram with compression-ratio-aware data encoding," in *2017 IEEE International Conference on Computer Design (ICCD)*. IEEE, 2017, pp. 573–580.

[88] D. Xue, C. Li, L. Huang, C. Wu, and T. Li, "Adaptive memory fusion: Towards transparent, agile integration of persistent memory," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2018, pp. 324–335.

[89] B.-D. Yang, J.-E. Lee, J.-S. Kim, J. Cho, S.-Y. Lee, and B.-G. Yu, "A low power phase-change random access memory using a data-comparison write scheme," in *2007 IEEE International Symposium on Circuits and Systems*. IEEE, 2007, pp. 3014–3017.

[90] H. Yu and Y. Du, "Increasing endurance and security of phase-change memory with multi-way wear-leveling," *IEEE Transactions on Computers*, vol. 63, no. 5, pp. 1157–1168, 2014.

[91] S. Yu, Y. Y. Chen, X. Guan, and H.-S. P. Wong, "A Monte Carlo Study of the Low Resistance State Retention of HfOx Based Resistive Switching Memory," *Applied Physics Letters*, 2012.

[92] J. Yue and Y. Zhu, "Accelerating write by exploiting pcm asymmetries," in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2013, pp. 282–293.

[93] F. Zahoor, T. Z. A. Zulkifli, and F. A. Khanday, "Resistive random access memory (rram): an overview of materials, switching mechanism, performance, multilevel cell (mlc) storage, modeling, and applications," *Nanoscale research letters*, vol. 15, no. 1, pp. 1–26, 2020.

[94] D. Zhang, L. Ju, M. Zhao, X. Gao, and Z. Jia, "Write-back aware shared last-level cache management for hybrid main memory," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2016, pp. 1–6.

[95] J. Zhang, N. Beckwith, and J. J. Li, "Gordon: Benchmarking optane dc persistent memory modules on fpgas," in *2021 IEEE 29th Annual*

International Symposium on Field-Programmable Custom Computing Machines (FCCM), 2021, pp. 97–105.

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

[97] M. Zhang, L. Zhang, L. Jiang, F. T. Chong, and Z. Liu, "Quick-and-dirty: An architecture for high-performance temporary short writes in mlc pcm," *IEEE Transactions on Computers*, vol. 68, no. 9, pp. 1365–1375, 2019.

[98] M. Zhang, L. Zhang, L. Jiang, Z. Liu, and F. T. Chong, "Balancing Performance and Lifetime of MLC PCM by Using a Region Retention Monitor," in *International Symposium on High Performance Computer Architecture*, 2017.

[99] W. Zhang and T. Li, "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, 2009, pp. 101–112.

[100] Y. Zhang, D. Feng, W. Tong, J. Liu, C. Wang, and J. Xu, "Tiered-reram: A low latency and energy efficient tlc crossbar reram architecture," in *2019 35th Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 2019, pp. 92–102.

[101] Y. Zhang, D. Feng, W. Tong, J. Liu, C. Wang, and J. Xu, "Tiered-reram: A low latency and energy efficient tlc crossbar reram architecture," in *2019 35th Symposium on Mass Storage Systems and Technologies (MSST)*, 2019, pp. 92–102.

[102] M. Zhao, H. Wu, B. Gao, X. Sun, Y. Liu, P. Yao, Y. Xi, X. Li, Q. Zhang, K. Wang *et al.*, "Characterizing endurance degradation of incremental switching in analog rram for neuromorphic systems," in *2018 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2018, pp. 20–2.

[103] M. Zhou, Y. Du, B. Childers, R. Melhem, and D. Mossé, "Writeback-aware partitioning and replacement for last-level caches in phase change main memory systems," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 8, no. 4, pp. 1–21, 2012.

[104] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," *ACM SIGARCH computer architecture news*, vol. 37, no. 3, pp. 14–23, 2009.