LADDER: Architecting Content and Location-aware Writes for Crossbar Resistive Memories

Md Hafizul Islam Chowdhuryy
Department of ECE
University of Central Florida
reyad@knights.ucf.edu

Muhammad R. Haq Rashed
Department of ECE
University of Central Florida
rashed09@knights.ucf.edu

Amro Awad
Department of ECE
North Carolina State University
ajawad@ncsu.edu

Rickard Ewetz
Department of ECE
University of Central Florida
rickard.ewetz@ucf.edu

Fan Yao
Department of ECE
University of Central Florida
fan.yao@ucf.edu

ABSTRACT

Resistive memories (ReRAM) organized in the form of crossbars are promising for main memory integration. While offering high cell density, crossbar-based ReRAMs suffer from *variable* write latency requirement for RESET operations due to the varying impact of IR drop, which jointly depends on *the data pattern of the crossbar* and *the location of target cells being RESET*. The exacerbated worst-case RESET latencies can significantly limit system performance.

In this paper, we propose **LADDER**, an effective and low-cost processor-side framework that performs writes with variable latency by exploiting both content and location dependencies. To enable content awareness, LADDER incorporates a novel scheme that maintains metadata for per-row data pattern (i.e., number of 1's) in memory, and performs efficient metadata management and caching through the memory controller. LADDER does not require hardware changes to the ReRAM chip. We design several optimizations that further boost the performance of LADDER, including LRS-metadata estimation *that eliminates stale memory block reads*, intra-line bit-level shifting *that reduces the worst-case LRS-counter values* and multi-granularity LRS-metadata design *that optimizes the number of counters to maintain*. We evaluate the efficacy of LADDER using 16 single- and multi-programmed workloads. Our results show that LADDER exhibits on average 46% performance improvement as compared to a baseline scheme and up to 33% over state-of-the-art designs. Furthermore, LADDER achieves 28.8% average dynamic memory energy saving compared to the existing architecture schemes and has less than 3% impact on device lifetime.

CCS CONCEPTS

• **Hardware → Non-volatile memory**; Emerging architectures; • **Computer systems organization → Processors and memory architectures**.

MICRO '21, October 18–22, 2021, Virtual Event, Greece

ACM ISBN 978-1-4503-8557-2/21/10…$15.00
https://doi.org/10.1145/3466752.3480054

KEYWORDS

Non-volatile Memory, Crossbar ReRAM, RESET Latency, Architecture Support, Performance Optimization, Metadata Management

ACM Reference Format:
Md Hafizul Islam Chowdhuryy, Muhammad R. Haq Rashed, Amro Awad, Rickard Ewetz, and Fan Yao. 2021. LADDER: Architecting Content and Location-aware Writes for Crossbar Resistive Memories. In *MICRO'21: 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '21), October 18–22, 2021, Virtual Event, Greece.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3466752.3480054

1 INTRODUCTION

The increasing trend for data intensive and high performance computing applications has significantly pushed the demand for efficient and scalable memory systems [16]. Due to the poor scalability and energy inefficiency of DRAMs, there has been a continuous effort to search for alternative memory technologies [4, 5, 29, 42, 45, 47]. Emerging non-volatile memories (NVMs) are promising contenders to augment or replace DRAM as they offer high density, energy efficiency, and non-volatility [27, 41]. Resistive random access memory (ReRAM) is particularly attractive for main memory integration due to its advantageous characteristics, including higher cell density and better data retention ([35, 53]) compared to other NVM devices such as phase-change memory (PCM) and spin-transfer torque MRAM (STT-MRAM) [30]. When arranged in a crossbar architecture [52], ReRAM can obtain the theoretically smallest cell size, which yields superior density and scalability.

Unfortunately, there are still outstanding challenges to be addressed for the adoption of NVMs. Particularly, NVM devices typically exhibit poor write performance [9, 26, 38]. This is especially the case for crossbar ReRAM memories for RESET operations, i.e., switching cells from high-resistive state (HRS) to low-resistive state (LRS). When accessing cells in the crossbars, the access latency can be elongated by the IR-drop over the array parasitics. In fact, the effective voltage drop across a ReRAM cell and thereby the access latency are heavily influenced by both *the data pattern stored in the crossbar* and *the location of the selected cells*. This leads to highly variable RESET latency requirements for ReRAM writes [10]. Employing the worst-case RESET latency could tremendously degrade the write performance since the RESET operation can be prolonged up to 10× [18] under a slight reduction in effective voltage drop (i.e., 0.4V). Although writes are generally not in the critical path

of program execution, long ongoing writes could block memory reads, which adversely impacts system performance [38].

To mitigate the write performance issue, circuit-level techniques such as applying ground bias and inserting non-linear selector devices are proposed to limit the sneak current in crossbars [48, 52, 61, 66]. Recent work in [68] employs dynamic reset voltage to compensate the IR-drop based on the location of target cells. Meanwhile, architectural techniques are designed to enable scheduling of writes with different RESET latencies [50, 52, 62]. Note that these techniques either do not jointly exploit the data-location dependencies (therefore not harnessing the full potential) or require additional circuitry supports (e.g., profiling functionality) in ReRAM devices that could increase the memory design complexity. As area and cost are the top design constraints for memories, it is desirable to come up with solutions that enhance ReRAM write performance without posing unnecessary hardware burden to the memory subsystem.

In this paper, we propose a novel framework–LADDER–that harnesses both data and location dependencies to enable multi-tiered write latency for ReRAM devices. To realize content-awareness, LADDER integrates a lightweight architectural scheme that maintains per-wordline *LRS-metadata* to record the number of '1's in each row. We build an accurate latency model for the ReRAM RESET operations by jointly modeling content (i.e., LRS cells along the wordlines) and the target locations (i.e., in terms of both wordline and bitline) for writes. With such knowledge, LADDER dynamically determines the varying but sufficient timings as writes are persisted. In LADDER, the memory controller stores LRS-metadata in a small reserved region of the main memory and manages them via regular read/write interfaces, thus *it does not require any change to the ReRAM chip or memory commands*. We further design a novel LRS-metadata caching mechanism to minimize the runtime overhead due to metadata accesses. Finally, we develop several optimizations to significantly improve system performance and efficiency of LADDER: (i) We propose an effective *metadata approximation technique* that gets rid of stale memory block reads for metadata updates; (ii) We deploy an *intra-line bit-level shifting technique* to balance the data pattern in the worst-case bytes among the wordlines in which the data line is mapped to, which further optimizes the required write latency; and (iii) We utilize a multi-granularity LRS-metadata design using different counter precision to reduce the metadata maintenance cost. We implement LADDER and evaluate it with full-system simulations using workloads from SPEC2006 and PARSEC benchmarks [7, 19]. Results show that LADDER achieves 46% performance gain over the baseline scheme using a pessimistic fixed RESET latency, and up to 33% speedup compared to prior works in [50] and [52]. Furthermore, LADDER achieves 53% and 28.8% dynamic energy savings in memory over the baseline and state-of-the-art designs exploiting bitline data patterns [50], respectively. Lastly, LADDER introduces only 3% additional writes on average to maintain LRS-metadata and has less than 1% memory storage overhead after applying the optimizations. By integrating LADDER with existing wear-leveling techniques [39, 59, 67], LADDER achieves almost the same lifetime as the baseline (i.e., only 2.9% reduction). In summary, the contributions of our work are:

- We propose LADDER, an efficient processor-side architecture support that enables ReRAM writes using varying latencies by modeling content and location dependencies.

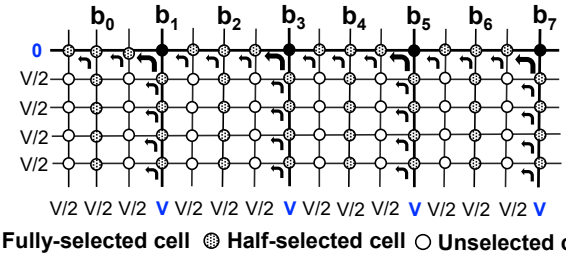


Figure 1: Illustration of the write operation in ReRAM based crossbar array where b_1, b_3, b_5, b_7 are under RESET.

- We design an LRS-metadata management and caching scheme that maintains the per-row data patterns in the memory controller and propose several enhancement techniques to significantly boost system performance of LADDER.
- We implement a prototype of LADDER and evaluate its efficacy with representative single- and multi-programmed workloads. Evaluations show LADDER achieves promising improvements in performance and energy savings with insignificant hardware cost.
- We present the ways existing wear-leveling techniques can be integrated with LADDER and their impacts on system performance and lifetime. We also offer several discussions about our LADDER design, including the crash consistency issue for LRS-metadata and the potential solutions.

2 BACKGROUND AND MOTIVATION

2.1 ReRAM Basics

ReRAM cells. While many types of resistive memory exist, a ReRAM cell is typically built using metal and oxide layers that enable variable resistance [64]. Each cell can either be in low-resistance state (*LRS*, logical bit '1') or high-resistance state (*HRS*, logical bit '0'). To read a ReRAM cell, a small voltage is applied across the device and the output current is measured. The write operation is performed by applying a write voltage under certain polarity, magnitude and duration. Typically, switching a ReRAM cell from LRS state to HRS state is called *RESET* operation, and switching from HRS to LRS state is called *SET* operation.

ReRAM crossbar array and memory organization. The cells in ReRAM are typically organized in dense array structures. Such structure allows sharing of many peripheral circuitry that offers high area efficiency. Each memory cell can be constructed using either a combination of access transistor and ReRAM cell (i.e., 1T1R), a selector-accessed ReRAM cell (i.e., 1S1R) or an access-free ReRAM cell (i.e., 0T1R) [35, 50, 52]. Typically, access-free and selector-accessed crossbar arrays have the highest area efficiency where the memory cell can achieve the theoretical minimum area of $4F^2$. The ReRAM cells then form mats (e.g., 512×512) to support read and write operations. One ReRAM memory module is organized as a hierarchy of ranks, chips and banks. Each bank is built using rows and columns of the ReRAM-based mats.

Variable latency requirement for ReRAM write. A ReRAM write operation is typically split into a *RESET* followed by a *SET*

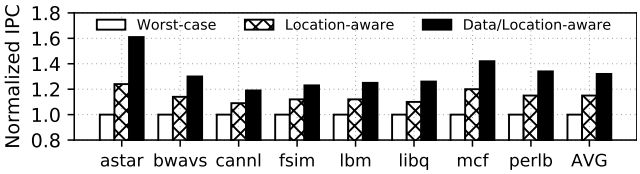


Figure 2: Potential system performance gain using location and content dependencies for writes in crossbar ReRAM.

operation [52]. As shown in Figure 1, the RESET operation is performed by grounding the wordline of the targeted cells and applying a voltage V to the target bitlines. The remaining wordlines and bitlines are then given a voltage of V/2 to minimize the write disturbance. Note that the SET operation is done in a similar fashion with the difference in reversing the voltage applied for the target wordline and bitline. During the RESET operation, the wordlines and bitlines of the target cells are selected. The ReRAM cells in the crossbar can have three different states: *fully-selected*, *half-selected* and *unselected*. The fully-selected cells are the targeted cells being written to. The cells that are along the selected wordlines and bitlines are half-selected cells, while the rest of the cells in the crossbar are unselected. Importantly, the voltage drop across the target cells is reduced because of the IR drop in the crossbar array [60]. The IR drop is introduced by the sneak current flowing through the *non-zero resistance* of the wordlines/bitlines as well the *half-selected cells* (See Figure 1). It has been observed that the RESET latency of a ReRAM cell is exponentially proportional to the voltage drop across the target cell, which can be modeled as $t = C \cdot e^{-k|V_d|}, k \geq 0$, where t is the latency of RESET operation, V_d is the voltage drop across the selected cell and k, C are constants [61].

To mitigate the IR drop issue for RESET, circuit-level techniques such as inserting highly non-linear selectors and applying voltage and group biasing (e.g., DSGB [52] and DSWD [65]) have been studied. However, these techniques only partially alleviate the IR drop issues. Recently, Zokaee et al. proposed a dynamic RESET voltage regulation (DRVR) and partition reset (PR) scheme that compensates the IR drop by supplying higher voltages and having extra RESETs and SETs for certain ReRAM cells [68]. Although this approach can bring down the worst-case RESET latency, it increases the complexity of the charge pump design [52] in ReRAM devices and also introduces non-trivial degradation of energy efficiency and lifetime as compared to the fixed-voltage mechanisms. Furthermore, supplying higher voltage can introduce write disturbance near the write driver [34]. Due to the cost-sensitive nature of commodity memory, it is desirable to have an architectural scheme that takes advantage of the RESET latency asymmetry to improve system performance without adversely impacting the complexity, performance, and energy envelope of ReRAM devices.

2.2 Exploiting Content and Location Dependency

We note that a significant variation in RESET latency exists across the ReRAM crossbar. Specifically, ReRAM cells that are closer to the write drivers suffer less from IR drop as compared to cells sitting in the far-end of the crossbar. On the other hand, for cells to be written in the same location, the IR drop can vary further by *the number of half-selected cells that are in LRS state*. Essentially, higher percentage of LRS cells in the selected wordlines and bitlines leads to greater IR drop. While applying the worst-case RESET latency guarantees correct and reliable operation of ReRAM devices, it will leave an excessive margin for write latency among the vast majority of ReRAM cells in the crossbar. If writes are performed using only the *minimally required* RESET latency values, considerable performance improvement may potentially be achieved. Figure 2 demonstrates the performance potential under ideal write schemes where content and/or location dependencies are known for RESET latency determination (See Section 5 for more details). As we can see, compared to the baseline scheme where the worst-case RESET latency is used for all writes, writes with location-aware RESET latency achieves maximum 24% IPC improvement. More notably, by modeling both content and location dependencies, the write scheme gains more than **1.6**× speedup. This clearly indicates the performance advantage of enabling content- and location-aware writes in ReRAM memory. Note that as long as the duration for RESET operation is higher than the *minimally required time*, the ReRAM device can perform writes correctly without introducing stability issues [50, 53, 68]. However, realizing such performance advantage is challenging as data pattern in the crossbar is by default unknown to the memory controller. Our work focuses on designing an efficient and low-overhead processor side mechanism that exploits the variant RESET latency to optimize system performance.

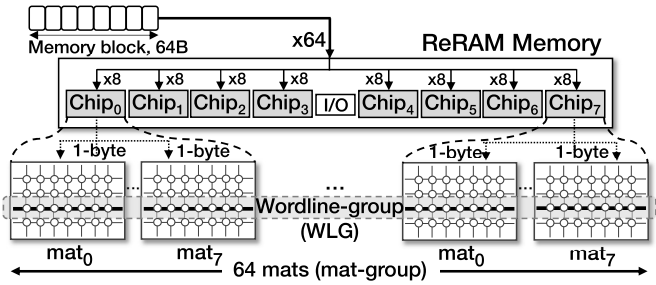
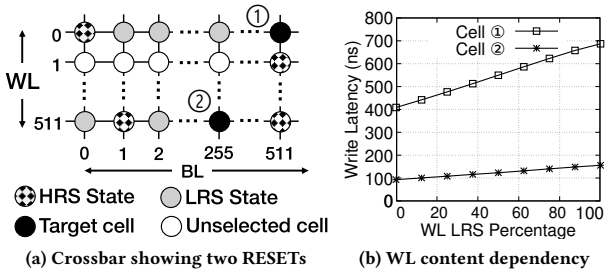


Figure 3: Physical mapping of a memory block in memory.

3 LADDER FRAMEWORK

3.1 Modeling Data and Location Dependency for Writes

We assume that the memory module utilizes ×8 ReRAM chip with mat size of 512×512, which provides the best balance between performance and reliability [34]. Typically, a 64B data line is mapped to 64 ReRAM mats across 8 chips [50, 52, 68]. Each chip stores 8 bytes to 8 individual mats and one byte is written to a specific wordline of a mat (as shown in Figure 3). We define these 64 mats and 64 wordlines as a *mat group* and a *wordline group* (**WLG**), respectively. Note that one WLG stores multiple memory blocks (with bytes spreading apart). We have already established that ReRAM cell RESET latency depends on two dimensions: 1) content of the neighboring cells and 2) location of the cells being RESET (i.e., the wordline and bitline indices). The location of a write can be determined by the memory controller using the memory addressing scheme. For content dependency, the half-selected cells in LRS state contribute to the sneak current (shown in Figure 1). Ideally, to fully



(a) Crossbar showing two RESETs (b) WL content dependency

Figure 4: WL-content dependency for the RESET latency. (a) Showing two RESETs to different locations; (b) RESET latency as a function of WL LRS cell percentage.

exploit content dependency, the numbers of LRS cells (logical '1's) among the 64 wordlines and 512 bitlines (for each of the 64 mats) are required to determine the minimal RESET latency.

Unfortunately, obtaining such knowledge from the memory controller is challenging. Reading all memory blocks in one WLG to derive the data pattern is clearly impractical due to the prohibitive performance overhead. Recent work in [50] proposes using profiling circuitry to track *bitline data patterns*. This approach relies on specialized circuit support and requires modifications to the memory-access protocol. Differently, as memory controller sees all data writes persisted to the main memory, it can keep track of numbers of '1's in WLGs on the fly. We call these counters *LRS-metadata*. The LRS-metadata can be loaded dynamically to determine RESET timing. Keeping LRS-metadata for both wordlines and bitlines can be prohibitively expensive as each data write will involve access to counters corresponding to 64 wordlines and 512 bitlines (Figure 3). To achieve a trade-off between overhead of LRS-metadata design and RESET timing precision, we build the write latency model where LRS-metadata is maintained for wordlines-only while assuming the worst-case data pattern for bitlines. To determine the proper write latency, the model uses a three-element tuple $\langle \mathbf{WL}, \mathbf{BL}, \mathbf{C^w_{lrs}} \rangle$ where **WL** and **BL** represent the location of wordline and bitline. $\mathbf{C^w_{lrs}}$ denotes the maximum number of LRS cells among the 64 wordlines where the data line is mapped to. We perform comprehensive circuit-level simulations to generate the data-location write latency model (i.e., a *three-dimensional* write timing table logically). Figure 4 demonstrates the changes of RESET latency as the wordline content varies for the two examples of write locations (See Section 5 for details of the latency model).

3.2 Enabling Variant-latency Write

We first discuss the major challenges and design considerations for enabling the variant-latency write mechanism. Figure 5 illustrates a high-level overview of a potential LRS-metadata scheme that realizes variant-latency writes. To harness the content-location dependency, the memory controller maintains lookup tables based on the derived latency model. The ReRAM main memory is logically divided into two regions storing regular data and the LRS-metadata. An addressing scheme can be built to determine data line address to LRS-metadata line address mapping. When data write arrives, the memory controller first generates the address of its corresponding LRS-metadata line. It will then read the metadata from the main

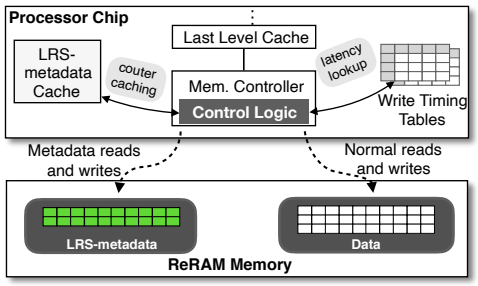


Figure 5: High-level framework for variant-latency write.

memory. To minimize LRS-metadata writes to ReRAM, LADDER integrates an LRS-metadata cache to temporarily store metadata memory blocks. The C^w_{lrs} derived from the LRS-metadata together with the location of the write (WL, BL) are used to lookup the write timing tables and determine necessary write latency. Since data write changes the content of the corresponding wordlines, the LRS-metadata must be updated. All metadata management operations are done by normal read and write operations to the ReRAM memory. We note that several design goals need to be accomplished to implement a practical variant-latency write scheme: (i) Efficiently encoding and storing LRS-metadata to reduce storage overhead; (ii) Minimizing the complexity of runtime LRS-metadata management; and (iii) Effectively caching metadata on-chip to mitigate energy and lifetime impact of counter maintenance.

3.3 Basic LADDER Design

In this section, we present a basic framework that would serve as the foundation for LADDER.

Accurate LRS cells counting. In order to determine the write latency for a data line, it is necessary to derive the maximum number of LRS cells among wordlines within the *wordline group*. One plausible way is to maintain counters that count the number of '1's in each of the wordlines, which we term as *LRS-counter*. Accordingly, 64 LRS-counters (i.e., $\mathbf{C^0_{lrs}} - \mathbf{C^{63}_{lrs}}$ will be used to derive $\mathbf{C^w_{lrs}}$ for one 64B data line write. We call this set of LRS-counter an *LRS-counter group*. Apparently, to ensure correct write operation, the RESET latency should be determined by the wordline that has the worst data pattern, i.e., $\mathbf{C^w_{lrs}} = \max \mathbf{C^i_{lrs}}$. The LRS-metadata essentially consists of LRS-counters grouped based on WLGs.

LRS-metadata storage and addressing. To minimize LRS-counter memory access for each memory write, it is beneficial to store the 64 counters in the *LRS-counter group* together so that the least number of metadata line reads are needed. With accurate counting, it takes 10 bits to represent the full range of each LRS-counter (0-512). Thus, one *LRS-counter group* consumes 80 bytes, which span two memory blocks. Note that since every wordline stores one byte from each individual memory block, the same LRS-counter group would be used for determining write latency for 64 memory blocks mapped to the same WLG. These memory blocks either correspond to one 4KB physical page or two half pages if channel interleaving is enabled. Under such scheme, the LRS-counter for the metadata block itself is not maintained. Therefore, to perform writes to LRS-metadata lines, we downgrade the latency model to be only location-dependent, essentially assuming the worst-case data pattern for writing metadata

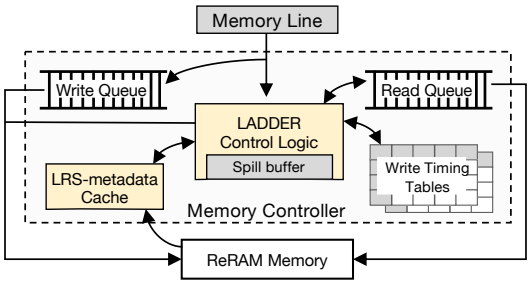


Figure 6: LADDER data write mechanism in basic design.

blocks. To store the LRS-metadata, the host system will pre-allocate a physical range of the main memory during boot time. This region is reserved for LRS-counter management and would not be exposed to the operating system for normal operations.

LRS-metadata update mechanism. To accurately update the LRS-metadata for data write, it is required to know the bits modified per wordline. By default, the memory controller is not aware of the original content upon receiving the request of writing a dirty line. To acquire such information, when a write request is inserted into the write queue, we augment the memory controller so that it will issue a read of the *stale memory block* (SMB) in the main memory. This SMB will be used to compare against the number of 1's (LRS cell) in the to-be-written cache line to provide the changes in each LRS-counter. The write queue entries are extended to allocate space for both the SMB and an additional *Present* (*P*) flag to indicate the existence of its LRS-metadata line in cache. We change the memory request scheduling algorithm to prioritize write requests with both SMB and counter line ready. When a write is dispatched from the write queue and written back to memory, the corresponding LRS-metadata is updated.

LRS-metadata caching. We propose a caching mechanism for LRS-metadata aiming to minimize the performance overhead of metadata accesses. An LRS-metadata cache is added in the processor to store active metadata lines used in the system. We leverage a Sharer (*S*) field in each tag entry that tracks the number of write requests (in write queue) whose to-be-written data blocks require this metadata line. When a conflict occurs in LRS-metadata cache, *S* is checked to evict an LRS-metadata line that is not used by any write queue entry at that time. In case all metadata lines in the conflicted set have non-zero sharer, the memory controller temporarily holds this request in a small *spill buffer*, and the metadata read would be issued once one of the lines in the corresponding set can be evicted. This checking is performed when the scheduler is switching from write to read mode.

Figure 6 illustrates the overview of the basic mechanism for serving writes with variable latency. We add a logic component called LADDER control logic that maintains metadata and derives write latencies. When a cache line write request is placed in the write queue, LADDER control logic first checks if the LRS-metadata is present in the LRS-metadata cache. If present, the *S* field of the corresponding LRS-metadata is incremented; otherwise, a metadata line read request is issued, along with the SMB read request. In this way, the read latency for metadata and SMB are largely overlapped with the request queuing time. Each read queue entry is augmented with a flag to indicate the type of read requests (namely *data*, *metadata* and *SMB*). When data response arrives at the memory controller, it is forwarded to the *Response Queue* in case of regular data, to the *Write Queue* for SMB, and to the *LRS-metadata cache* for metadata. When the data write is being serviced, LADDER logic first determines the required latency by constructing $\langle \mathbf{WL}, \mathbf{BL}, \mathbf{C^w_{lrs}} \rangle$ and checking the write timing tables. Then the LRS-metadata line is updated by LADDER logic based on the delta of the number of '1's in the current data and the stale memory block. Finally, the **S** field for the corresponding LRS-metadata line is decremented.

Employing Flip-N-Write (FNW) with LADDER. FNW [11] is a write-reducing mechanism widely deployed in NVM devices. Particularly, FNW chooses either the original memory bytes or the inverted bytes to write based on which one yields the least number of bit changes (including both SETs and RESETs). Note that the classical FNW scheme may invalidate the counting mechanism in LADDER since it can favor flipping that will decrease the total bit changes but incur more '1's to be written as compared to the original data. To be compatible with FNW, LADDER introduces a slight change in FNW operation by adding an additional constraint that the number of '1's in the variant written to memory cannot be higher than that of the original (i.e., *non-flipped*) memory block. We note that the corresponding hardware cost is minimal. Since FNW mechanism is typically implemented in a separate bridge chip, this does not involve any change in the ReRAM chips [24].

4 LADDER OPTIMIZATIONS

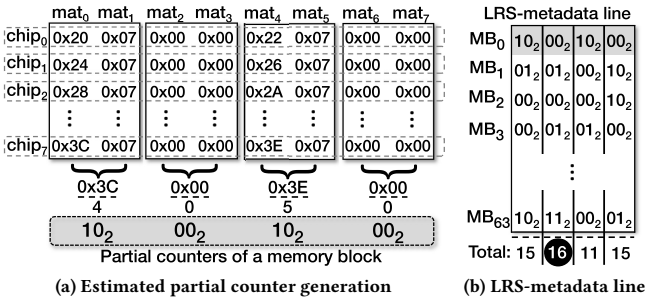
The basic LADDER design in Section 3 presents a framework that enables the memory controller to issue writes with variant latency. However, it can be subject to performance issues due to the SMB reads, and LRS-counter reads/writes. In this section, we propose several enhanced LADDER schemes to further boost performance.

4.1 LRS-metadata Estimation

The extra read of the stale memory block for each data write can contend with normal data reads that impact system performance. To tackle this issue, we propose an effective counter estimation technique that can *avoid all the SMB reads*. The key observation is that to determine the RESET latency for a cache line write, *only the maximum LRS-counter among its LRS-counter group* (i.e., $\mathbf{C^w_{lrs}}$) is needed. Let S_i^j denotes the number of '1's (or LRS cells) in the j^{th} byte of memory block i. For the wordline within its WLG with the maximum LRS cells (i.e., $\mathbf{C^w_{lrs}}$), assume S_i^x denotes the number of '1's in its 1-byte part from the i^{th} memory block. Then we have $C^w_{lrs} = \sum_{0 \le i < 64} S_i^x$. Since S_i^x is one of the 64B from memory block i, it will hold that $S_i^x \le \max_{0 \le j < 64} S_i^j$. If we denote $\max_{0 \le j < 64} S_i^j$ as S_i^M (i.e., the number of '1's in the *worst byte* of block i), we can derive the following inequality:

$$\mathbf{C^w_{lrs}} \le \sum S_i^M \tag{1}$$

This means that C^w_{lrs} can be bounded by the sum of the number of '1's of the 64 worst bytes, each from one of the 64 memory blocks. As a result, we can use $\sum S_i^M$ to estimate C^w_{lrs} while ensuring the derived latency is sufficient. We call S_i^M the *partial counter*. Note



(a) Estimated partial counter generation (b) LRS-metadata line

Figure 7: Counter estimation and organization. (a) An example of partial counters for a data line from the *astar* benchmark. Each hex number is one byte from a memory block mapping to a chip and one mat. Partial counters stored in the main memory are shown in the shaded block; (b) An illustration of partial counters in one LRS-metadata line. The circled value represents the C^{w}_{lrs} and the shaded block shows partial counters from the data line in (a).

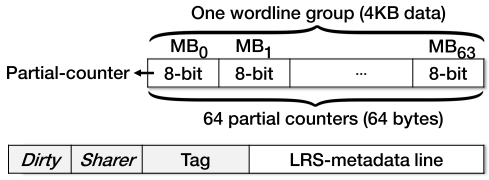


Figure 8: Storage of the partial counters in each LRS-metadata line (top); The LRS-metadata cache structure (bottom).

that this method may considerably overestimate the C^{w}_{lrs} value as the worst byte in each memory block may have a large number of '1's. We improve the estimation accuracy by dividing the mat group to N subgroups, correspondingly splitting each of the mapped memory blocks to multiple sub-blocks. We then keep the per-byte maximum number of '1's for block i in each subgroup, i.e., $S^{M_j}_{i}$ for j^{th} subgroup. Hence, under subgrouping, each memory block corresponds to N partial counters instead of one. Accordingly, the estimation based on Equation 1 is performed for each subgroup (i.e., $C^{w_j}_{lrs}$ for j^{th} subgroup). The final C^{w}_{lrs} can be derived based on the maximum of the counters among all subgroups using the inequalities:

$$C^{w_j}_{lrs} \leq \sum S^{M_j}_{i} \quad \text{and} \quad \mathbf{C^{w}_{lrs}} \leq \max\{C^{w_j}_{lrs}\} \tag{2}$$

Figure 7a demonstrates how LADDER generates the partial counters using memory blocks from *astar* as an example. We empirically set N to 4. Each WLG is divided into four subgroups. Thus, each memory block in the WLG has 16B mapped to the subgroup. The number of '1's in worst case byte is computed among these 16B. For the data line shown in Figure 7a, its partial counter values are $\langle 4, 0, 5, 0\rangle$. In practice, we use 2 bits to encode each partial counter for the range in $\langle 0..8\rangle$. Specifically, '00', '01', '10' and '11' represent partial counter values of 1 (range 0~1), 3 (range 2~3), 5 (range 4~5) and 8 (range 6~8), respectively. In this way, one byte can store all

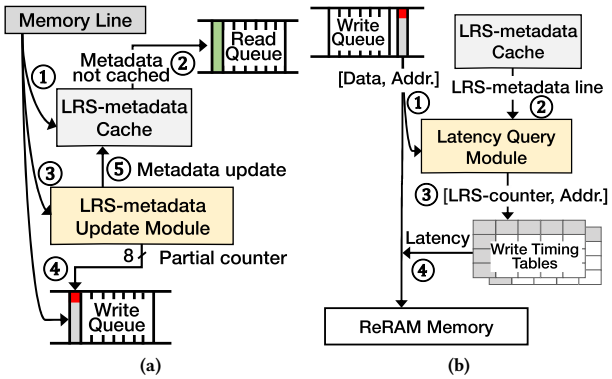
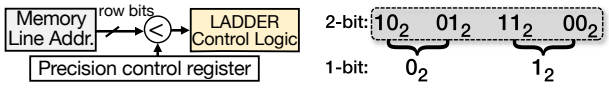


Figure 9: LRS-metadata estimation scheme. (a) Partial counter generation and LRS-metadata line update; (b) LRS-counter generation and the corresponding RESET latency determination.

four partial counters for one data line, and the partial counters in all data blocks in one WLG take 64B. Figure 7b shows the LRS-counter generation for the LRS-metadata line containing partial counters. The number (16) denotes the estimated worst-case number of '1's that would be used to determine write latency. Notably, there are two major advantages associated with this estimation technique: (i) When a dirty block is to be written, LADDER can update its metadata (i.e., partial counters) based on the worst-case byte in each sub-block of the data line. Therefore the stale memory block reads in the basic LADDER scheme could be *completely avoided*; and (ii) The encoding of partial counters enables LADDER to pack the counters for 4KB data (i.e., a physical page) to one memory block, as opposed to two blocks in the basic LADDER design (Section 3.2). This not only reduces the memory storage overhead for LRS-metadata but also improves the spatial locality of the metadata when writes are persisted. The top figure in Figure 8 shows the LRS-metadata line storage with partial counters. The LRS-metadata cache structure is shown at the bottom of Figure 8.

Figure 9 shows the optimized control logic in LADDER with LRS-metadata estimation. It contains two major components: an *LRS-metadata Update Module* and a *Latency Query Module*. As illustrated in Figure 9a, when a data write request arrives at the memory controller, LADDER checks the corresponding LRS-metadata line in the LRS-metadata cache (①) and issues an LRS-metadata line read in case of a cache miss (②). The LRS-metadata Update Module derives the partial counters for this data line following Equation 2 (③). It stores the 8-bit partial counters in the write queue along with the memory line (④), this will be used to update the LRS-metadata line when the data write is serviced by the memory controller (⑤). Figure 9b shows the RESET latency determination mechanism for a memory line write operation. When the write is being dispatched from the write queue, the Latency Query Module first generates the address of the LRS-metadata line corresponding to the current to-be-written data line (①). The Latency Query Module then retrieves the LRS-metadata line from the cache (②) and computes the LRS-counter based on the partial counters (as shown in Figure 7b). It then uses the generated LRS-counter to



(a) Multi-granularity LADDER ctrl. (b) Partial to 1-bit counters

Figure 10: The LADDER architecture with multi-granularity counters. (a) The counter granularity determination logic using a precision control register (8-bit); (b) 1-bit low-precision counter generation from partial counters.

Crossbar params.	Value	Crossbar params.	Value
Crossbar dimensions	512×512	Input resistance	100Ω
Number of selected cells	8	Output resistance	100Ω
ReRAM LRS resistance	$10K\Omega$	Wire resistance	2.5Ω
ReRAM HRS resistance	$2M\Omega$	Write voltage	$3V$
Selector non-linearity	200	Bias voltage	$1.5V$

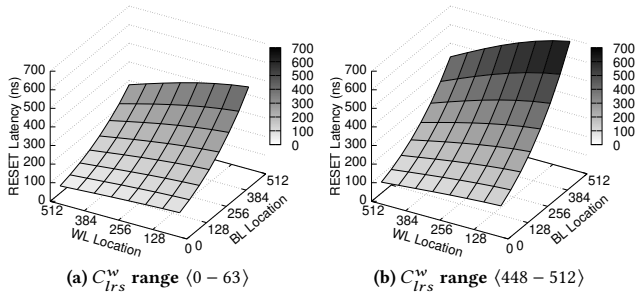
Table 1: ReRAM crossbar parameters.

query the corresponding write latency from *Write Timing Tables* (③). Finally, the memory line is written back to memory using the latency obtained (④).

Improving estimation performance with shifting. We observe that in many applications, the logic '1's are clustered together in a small number of mats. This pattern is consistent among consecutive data lines in a memory page. Similar observations about repetitive data patterns in a page were also made by prior works [59, 70]. This can lead to screwed $C_{lrs}^{w_k}$, which results in large partial counter values. To mitigate the impact of such phenomenon, LADDER performs intra-line bit-level shifting in data lines, so that clustered bytes with a large number of '1's are no longer packed in the same set of mats. For example, data pattern in Figure 7a shows clustered worst bytes in $mat_{\{0,1,4,5\}}$. We distribute these clustered '1's among mats in the same chip using bit-level shifting among the 8 bytes mapped to the same chip (e.g., the 8 bytes in each row in Figure 7a). To misalign data patterns across consecutive data lines, each data block in the WLG leverages a distinct shift offset based on its mapped position in the wordline. When a memory block is read from memory, a reverse shift operation is performed to recover the memory block in the original bit order.

4.2 Multi-Granularity LADDER Counters

Another source of overhead in the basic LADDER design is the additional reads and writes for metadata maintenance. Such overhead is especially pronounced when the LRS-metadata cache hit ratio is low. Notably, we observe that wordlines at the bottom of ReRAM crossbars are relatively insensitive to the per-row data patterns (See Figure 4b). This is because wordlines closer to the write driver experiences lower IR drop [62]. Consequently, *reducing the precision of counting* for the data blocks in bottom rows will unlikely incur a noticeable impact on write performance. Therefore, we propose a multi-granularity counter design where data blocks stored in bottom rows use two 1-bit partial counters (instead of 8-bit). Figure 10a illustrates the high-level design changes from LRS-metadata estimation scheme for multi-granularity counters. Specifically, we add a *precision control register* that contains an 8-bit mask representing



(a) C_{lrs}^{w} range $\langle 0-63 \rangle$ (b) C_{lrs}^{w} range $\langle 448-512 \rangle$

Figure 11: The derived RESET latency for writes at different WL/BL locations when the WL data pattern is (a) *all '0's* and (b) *all '1's.*

Hardware	Configurations
Processor	4-core, Out-of-order, x86
L1 I/D-Cache	Private, 32KB, 2-way
L2 Cache	Private, 4MB, 16-way
L3 Cache	Shared, 32MB, 16-way
Mem. Controller	32-entry **RDQ**; 64-entry **WRQ**, Write switching threshold: 85%
Counter Cache	64KB, 4-way, 2 cycles access latency
ReRAM Memory	16GB, dual channel, 2 ranks/channel, 8 banks/rank, 256 mats/bank, 512x512 crossbar
ReRAM Timing	**tCL:** 13.75ns, **tRCD:** 13.75ns, **tBURST:** 5ns, **tWR:** *29ns - 658ns*

Table 2: Architecture parameters in LADDER architecture.

the number of bottom rows with lower-precision LRS-metadata. An example of the 1-bit partial counter generation is shown in Figure 10b. In the 1-bit partial counter, value *0* represents a number of '1's within $\langle 0..5 \rangle$, and *1* denotes range $\langle 6..8 \rangle$. The *1-bit* partial counters enable even more compact metadata design where one 64-byte metadata block can store the LRS-metadata for *4 physical data pages* in bottom rows. This can improve LRS-metadata access locality as less LRS-metadata lines are accessed at runtime, leading to a reduced number of LRS-metadata reads/writes from memory.

5 EXPERIMENTAL SETUP

Circuit level simulation. To generate the latency model, we model the crossbar using the modified nodal analysis (MNA) [20]. The MNA equations are solved using sparse LU factorization together with backward and forward substitution. By exploiting the sparse structure of the crossbar, the simulation time is orders of magnitude faster than HSPICE while the *same accuracy* is maintained. We note that previous works have used different crossbar parameters [50, 52]. To provide a fair and well-organized comparison, we select one set of practical parameters based on prior works [25, 34, 50] and evaluate previous ReRAM write performance enhancing schemes under the same settings (See Table 1). Ideally, the most fine-grained latency model can specify a RESET latency based on exact BL/WL locations and WL data pattern (i.e., $512 \times 512 \times 512$). However, having such a fine-grained latency is impractical since the timing tables will consume tremendous on-chip storage. Furthermore, we

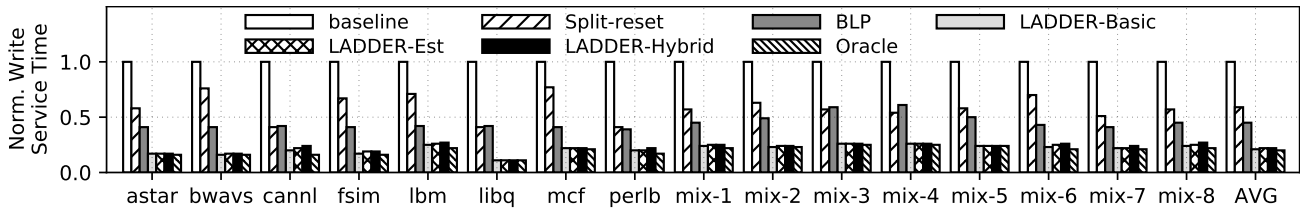


Figure 12: Average write service time to ReRAM memory.

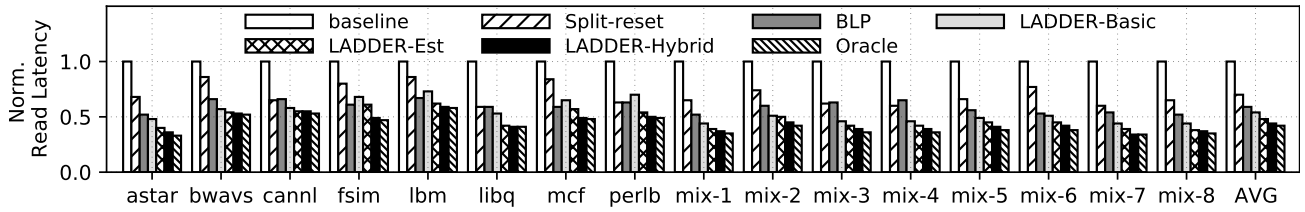


Figure 13: Average latency for processor data reads.

Single-program Workload	astar bwaves canneal facesim lbm libquantum mcf perlbench
Multi-program Workload	**mix-1**: astar-lbm-mcf-cactusADM **mix-2**: cactusADM-bwaves-perlbench-zeusmp **mix-3**: bwaves-zeusmp-astar-mcf **mix-4**: zeusmp-perlbench-lbm-cactusADM **mix-5**: cactusADM-astar-lbm-perlbench **mix-6**: zeusmp-cactusADM-bwaves-mcf **mix-7**: astar-lbm-bwaves-mcf **mix-8**: mcf-cactusADM-zeusmp-perlbench

Table 3: List of single- and multi-programmed workloads.

find that the latency difference is negligible for small changes in location and content, indicating that a highly fine-grained model is not necessary. In our evaluation, the timing table is logically organized as $8 \times 8 \times 8$ where each dimension (i.e., WL/BL/C^W_{lrs}) is set with a granularity of 64 (for 512×512 crossbar size). The memory controller maintains this timing model as 8 sub-tables (the C^W_{lrs} dimension) with the size of 8×8 (the BL and WL dimensions) to determine RESET latencies. Figure 11 illustrates the RESET latencies for writes at different write locations under two extreme WL data patterns (i.e., all '0's and all '1's) in our latency model, corresponding to two sub-tables (we do not enumerate all sub-tables due to space limit). Our investigation shows this reduced granularity has less than 3% impact on performance.

Architecture simulation configuration. We run experiments using gem5 [8] with full system simulation. By default, LADDER uses a 64KB 4-way set-associative LRS-metadata cache. We boot a Linux-based system with kernel version 3.4.112. Table 2 illustrates the main architectural parameters used in our LADDER framework. The write recovery latency `tWR` is varying according to content and location and is determined dynamically by LADDER.

Workloads. We configure 8 single-programmed benchmarks from SPEC2006 (with reference input) and PARSEC suite (with sim-large inputs) as well as 8 multi-programmed workloads (each is a mix of 4 SPEC2006 benchmarks) with high WPKI and large working sets [17] (shown in Table 3). We instruct each simulation to skip the first 5 billion instructions followed by 100M for cache warmup and then perform detailed simulations for half billion instructions.

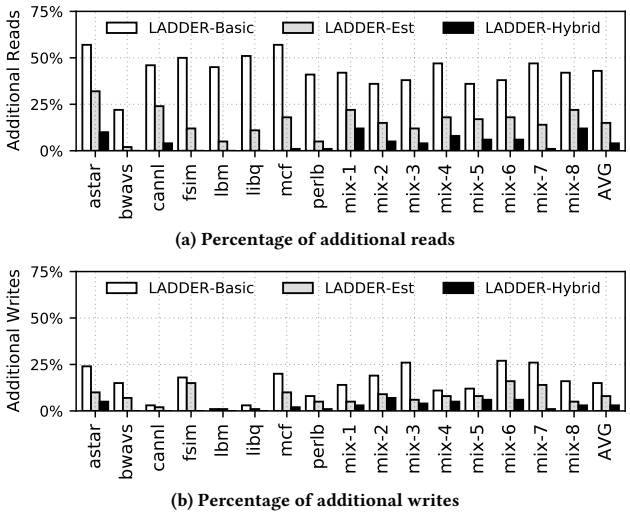
6 EVALUATION

6.1 Evaluation Methodology

We implement three variants of LADDER in gem5, namely LADDER-Basic that maintains accurate LRS-counter for wordlines, LADDER-Est that adopts the counter estimation and bit-level shuffling techniques, and LADDER-Hybrid that leverages multi-granularity counters. We empirically set the bottom 128 rows to use lower-precision LRS-metadata. LADDER schemes incorporate the FNW variant as discussed in Section 3.3. We empirically found that less than 4% of flipping operations are canceled due to the addition of the constraint that ensures the correctness of counting, which indicates the impact of FNW modification is minimal. We create a scheme utilizing worst-case data- and location-dependent latency as the first *baseline*. Additionally, we implement two state-of-the-art write performance enhancing techniques that are most related to LADDER: 1) **BLP** that harnesses bitline-level data pattern obtained from embedded profiling circuitry to determine write latency [49, 50]; 2) **Split-reset** that reduces RESET latency by dividing one RESET phase into 2 possible half-RESETs, each half-RESET stage writes maximum 4 bits to one mat (instead of 8) [52]. **Split-reset** utilizes data line compression so that compressible lines only need one half-reset, leading to faster write. We apply the same circuit-level simulation parameters (as shown in Table 1) to generate the latency model for *Split-reset* and *BLP*. Finally, we implement the **Oracle** write scheme where the memory controller performs writes using our latency model assuming LRS-counters are known. This mechanism can demonstrate the theoretical performance gain for mechanisms exploiting the data-location aware latency model. We apply classical FNW [11] for *baseline*, *BLP* and *Split-reset*.

6.2 Performance Evaluation

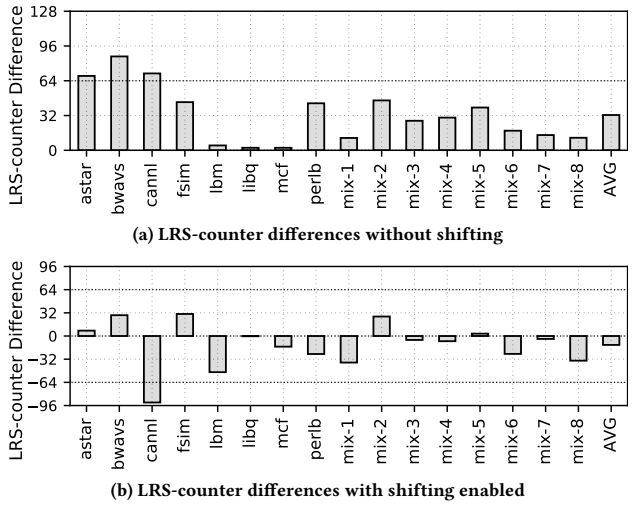
Read and write performance. We first evaluate the write and read performance for LADDER. As we can see from Figure 12, by enabling fast and slow writes, Split-reset achieves a 41% decrease in



(a) Percentage of additional reads

(b) Percentage of additional writes

Figure 14: Additional reads/writes in LADDER due to metadata maintenance (normalized to the total number of reads/writes in the *baseline*).



(a) LRS-counter differences without shifting

(b) LRS-counter differences with shifting enabled

Figure 15: The LRS-counter (C^w_{lrs}) differences (on average) between LADDER-Est and LADDER-Basic.

average write service time normalized to *baseline*. Additionally, BLP has 14% more reduction on top of Split-reset by modeling bitline data pattern. More notably, LADDER-Basic is able to achieve the best write performance achieving 79% decrease in write service time compared to *baseline*, while LADDER-Est and LADDER-Hybrid exhibit almost the same write performance gain as LADDER-Basic. We also observe that the counter estimation and multi-granularity counters are effective in deriving close-to-accurate RESET latencies (compared to Oracle). Figure 13 presents the average read latency, including both queuing time and servicing time for all schemes. We can see that LADDER consistently shows lower read latency. LADDER-Hybrid has the best read performance with 37% and 16% more latency reduction on top of *Split-reset* and *BLP*, respectively. Furthermore, both LADDER-Est and LADDER-Hybrid perform better than LADDER-Basic in terms of reads. This is because these two schemes dramatically mitigate interference with normal data reads by reducing additional SMB/metadata reads and additional LRS-metadata writes.

To better understand the usefulness of LADDER optimization schemes, we evaluate the additional read/write operations due to the LRS-metadata maintenance compared to *baseline*. Note that different from the other two LADDER schemes, LADDER-Basic's read overhead includes two components: SMB read and LRS-metadata read. Figure 14a shows that LADDER-Basic has additional reads averaging at 43% among all workloads. This is mainly contributed by the SMB reads for each data write. By avoiding the SMB reads, the read overhead is substantially lowered to 15% in LADDER-Est and then further lowered to 4% in LADDER-Hybrid by using a more compact layout of metadata that considerably improves metadata line locality. Figure 14b shows the additional writes for our LADDER schemes. Compared to the LADDER-Basic, LADDER-Est has less metadata writes (at 8% on average). This is because an LRS-metadata line in LADDER-Est can store counters for 4KB data blocks whereas LADDER-Basic requires two for the same. Finally, LADDER-Hybrid can further reduce it to only 3%.

Effectiveness of LRS-metadata estimation. To evaluate the accuracy of the counter estimation techniques, we run experiments that trace the accurate and estimated counters for every write persisted. Figure 15a and Figure 15b show the average counter value difference between LADDER-Basic (that uses accurate counting) and LADDER-Est (that performs counter estimation). When deployed without intra-line bit shifting, LADDER-Est has in general higher counter values compared to LADDER-Basic. However, only 3 out of the 16 benchmarks have average estimated counter differences above 64. Most of these estimation errors *do not result in a worsened* `tWR` compared to accurate counting since LADDER encodes data patterns using 8 levels for one wordline of 512 bits (See Section 5). Additionally, the LADDER-Est with bit shifting distributes the bits in a cacheline in a way that every bit in an 8-byte group (that comes from the same chip) gets distributed to the mat group. As shown in Figure 15a, LADDER-Est can actually have smaller LRS-counters compared to the LADDER-Basic for majority of the benchmarks, even after considering the effect of estimation.

System performance of LADDER schemes. We evaluate the overall system performance improvement under all schemes in Figure 16. Note that we use IPC for single-programmed workloads and weighted IPC [40] for the multi-programmed workloads to compute speedup. Compared to baseline, *Split-reset* has a performance improvement of 13% and 27% on average for single- and multi-programmed workloads respectively, whereas *BLP* has 22% and 27%. By leveraging a more aggressive latency model with efficient metadata management, LADDER schemes exhibit even higher performance gains. Specifically, LADDER-Basic can bring on average 22% IPC improvement for single workloads and 50% for multi-programmed workloads over baseline. Furthermore, LADDER-Est achieves 5% more IPC improvement for single-threaded workloads and 4.7% for mixed workloads on top of LADDER-Basic. The higher

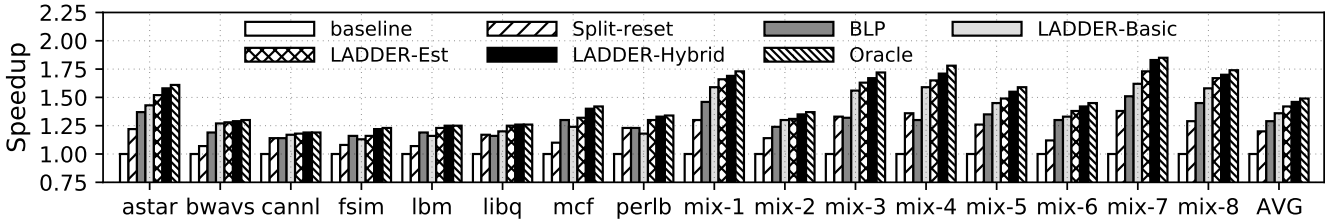


Figure 16: Comparison of performance (i.e., speedup normalized to baseline) under different schemes.

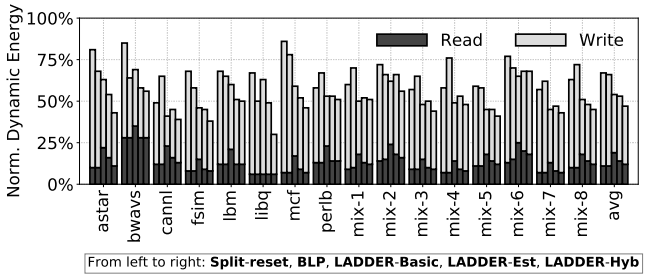


Figure 17: Memory dynamic energy consumption analysis.

performance advantage of LADDER-Est is attained due to the removal of SMB reads, which eliminates contentions with regular data reads. Finally, the LADDER-Hybrid scheme has additional 2.8% performance benefits over LADDER-Est. As compared to previous techniques, LADDER-Hybrid has 21.7% and 13.2% speedup over Split-reset and BLP. It is important to note that besides higher system performance, LADDER-Hybrid also limits the read and write overhead to less than 4% and 3% respectively. By utilizing a low-overhead architectural solution, LADDER can achieve 98% of the performance of the *Oracle* scheme.

6.3 Hardware Overhead Analysis

Main memory overhead analysis. We compare the dynamic energy consumption for ReRAM memory among the studied mechanisms. We model a ReRAM memory with the device-level parameters from [25] and use NVmain [37] to analyze the dynamic energy. Note that all the studied schemes save write energy compared to *baseline* due to the optimized write latency for data writes. Figure 17 shows the dynamic energy consumptions including *read* and *write* energy at mat level (normalized to *baseline*). We can see that Split-reset and BLP present an energy saving of 33% and 34%, respectively. Interestingly, for the benchmarks with high data line compressibility (e.g., *canneal* and *perlbench*), Split-reset has less dynamic energy consumption compared to BLP. This is because, for a compressed data line, Split-reset only requires one half-reset phase in contrast to the full-reset in BLP. LADDER-Basic reduces the memory dynamic energy by 46% over *baseline*. By further mitigating energy cost due to SMB/metadata reads, 48% and 53% energy is saved by LADDER-Est and LADDER-Hybrid. Impressively, LADDER-Hybrid has 29.8% and 28.8% less dynamic energy consumption compared to Split-reset and BLP.

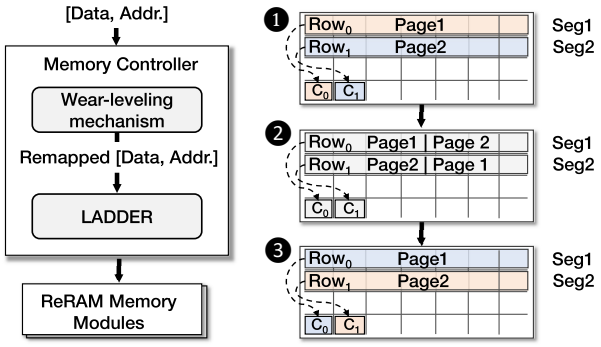
LADDER-Basic requires two 64-byte memory blocks to store counters for one logical row, which translates to 3.12% memory space for storing LRS-metadata. LADDER-Est reduces the storage overhead to 1.56% since this scheme only requires one 64-byte counter for every 4KB data. Finally, LADDER-Hybrid minimizes this overhead even further to 0.97% with coarse-grained partial counters for selected wordlines. Our proposed mechanisms essentially trade a minimal cost of ReRAM memory capacity for higher performance gains. NVM memories have significantly higher capacity compared to traditional DRAMs, such memory storage overhead is unlikely to have a noticeable impact on applications.

Module	Area (mm^2)	Power (mW)	Latency (ns)
LRS-metadata Update Module	0.0061	3.71	0.17
Latency Query Module	0.0047	6.57	0.32
LRS-metadata Cache (64KB)	0.2442	48.83	0.81

Table 4: Hardware overhead of LADDER.

Memory controller overhead analysis. To evaluate the hardware overhead in the memory controller, we implement the two major logic components (LRS-metadata Update Module and Latency Query Module) in the optimized LADDER schemes using Verilog. Note that the processor-side hardware overheads for LADDER-Est and LADDER-Hybrid are almost identical. We synthesize the proposed logic components using Synopsis Design Compiler with the 45nm FreePDK45 standard cell library [44]. The area, power and latency overheads are shown in Table 4. We observe that the LRS-metadata Update Module and Latency Query Module mechanisms have negligible area overhead (e.g., compared to the $263mm^2$ chip area of Intel Core i7 Processor [1]). Additionally, the dynamic power drawn by each component is only in the order of a few mWs. Furthermore, LADDER-Est hardware has latencies less than the processor clock cycle. Also, as LRS-metadata maintenance can be overlapped with the queuing of write requests, they do not pose additional latency overhead to the system. Finally, we use CACTI 7 [6] to model the 64KB 4-way LRS-metadata cache. The LRS-metadata cache incurs about 49mW dynamic power and $0.25mm^2$ area overhead, which are both modest numbers. Also, we observe marginal system performance gain when increasing cache size ($< 2\%$).

The LADDER write timing tables require a 512B buffer as on-chip storage. This timing information is loaded by the processor at boot time. Since the actual RESET timing may be memory device dependent, memory device manufacturers could choose to program the latency information into a ROM on the memory module [2] (e.g., *Serial Presence Detect* [2]) in memory DIMMs. Note that the JEDEC standard already has supports for additional configuration storage (e.g., model registers) to enable conveying device-level information to the system [3]. This approach has minimal cost and does not pose any additional complexity to the ReRAM chips. Prior



(a) Wear-leveling in LADDER (b) Segment-based wear-leveling

Figure 18: LADDER with wear-leveling mechanisms. (a) Control flow of the combined logic; (b) Procedure of LRS-metadata block remapping when $Segment_1$ is swapped with $Segment_2$.

works have proposed leveraging such storage to expose certain memory device information to the processor-side to enhance the performance of memory subsystems [28]. We use a 16-entry spill buffer, each entry storing the address of an LRS-metadata line. For LADDER-Est and LADDER-Hybrid, we need 8 bits for partial counters, 1 bit for the *Present* flag per write queue entry, and 1 bit for determining read operation type per read queue entry. In LADDER-Basic, we need additional 256 bits to store data line bit modification information instead of the partial counter. In summary, the on-chip storage is modest. Note that LADDER is a processor-side framework that does not require changes of ReRAM chip and the memory interfaces, retaining the memory area and cost efficiency for ReRAM technologies.

6.4 LADDER with Wear Leveling Techniques

Wear-leveling has been widely adopted to enhance the lifetime of NVM devices through distributing writes evenly across the entire memory region [39, 70]. Typically, vertical wear-leveling (VWL) [39, 43] manifests at a line or segment (e.g., 1MB) granularity while horizontal wear-leveling (HWL) [59, 67] distributes writes within a line. Note that LADDER can be easily made compatible with existing wear-leveling techniques by putting the wear-leveling operations before LADDER. As shown in Figure 18a, LADDER can obtain the LRS-metadata line based on the remapped physical location of the data block (for VWL) and update the LRS-metadata based on the data written back to memory (for HWL). It is worth noting that different implementations of VWL can have distinctive impacts on the performance of LADDER. Specifically, line-based wear-leveling can potentially distribute the data blocks within a page to different logical rows, leading to deteriorated LRS-metadata locality. On the other hand, segment-based VWL typically remaps memory in large chunks containing one or multiple pages (e.g., 16MB [67]). Under this mechanism, when a data page is relocated from one logical row to another, the metadata locality is maintained (mapping from one metadata line to another). Since content in the original location (before wear-leveling) remains unmodified, LADDER only needs to update the metadata line corresponding to the new location (shown in Figure 18b). Note that when the *metadata pages* are relocated, there is no impact on normal data writes as the memory controller can locate the remapped metadata lines. Finally, HWL generally shifts one byte at a time [67] in a memory block that does not change the address of the corresponding metadata line, thus no special handling is needed.

We analyze the endurance of a crossbar based on the cell with worst-case endurance [39, 63, 68]. Due to the variable IR-drop across crossbars, different cells have different endurance. Note that LADDER does not change the endurance of ReRAM memories from baseline as it maintains the same RESET voltage. According to Figure 14b, LADDER-Hybrid increases write traffic by 3% compared to the baseline. By applying wear-leveling schemes, the impact of these additional writes in our LADDER framework can be distributed over the entire crossbar. With the adoption of such techniques, LADDER-Hybrid maintains 97.1% of the baseline system lifetime. In such a system, LADDER-Hybrid has only about 1% performance reduction (compared to that of a system without wear-leveling), still achieving 44% improvement over baseline. LADDER-Basic and LADDER-Est exhibit about 2% overhead for the same.

7 DISCUSSION

Crash consistency of LRS-metadata. Regular data writes are typically guaranteed to persist to the non-volatile main memory. However, as dirty LRS-metadata blocks can be temporarily stored in the LRS-metadata cache, it is possible that they are not properly flushed upon abrupt power failure. Under such circumstances, the LRS-metadata loaded from memory after power restoration can be obsolete, leading to inconsistency. The use of stale LRS-counters to derive RESET latency may result in write failures for ReRAM devices. Recent commercial processors offer eADR mechanisms that encompass on-chip caches into the persistence domain [21], which could provide crash consistency if extended to the LRS-metadata cache. In systems where such mechanism is absent, one plausible solution is to perform *Lazy LRS-metadata correction*. Specifically, when restoring from a crash, the system can conservatively overwrite the LRS-metadata with maximum values in the metadata region (less than 1% of the memory space). This ensures that later data writes will use *safe* RESET timings if the corresponding LRS-metadata were not persisted. As memory blocks that share the same LRS-metadata are written, their LRS-counters will be gradually adjusted to the expected estimations. Note that since crashes are infrequent and the correction happens only at the initial stage of the system, we do not expect a noticeable long-term performance impact with this scheme. Additionally, we note that prior works [58, 69] have proposed techniques to identify inconsistent metadata in secure NVMs by detecting/bookkeeping non-persistent metadata blocks. LADDER can integrate such approaches to locate only stale LRS-metadata blocks for conservative correction, which could further reduce the overhead of system restoration.

Write reliability of LADDER and its optimizations. LADDER's latency model is built using comprehensive circuit-level analysis using ReRAM parameters that match the prototype ReRAM devices (Table 1). We note that the reliability of writes is maintained in LADDER for two reasons: i) the timing model specifies varying but *sufficient* RESET latency to switch the resistive states of the

ReRAM cells based on the location and data pattern; and ii) LADDER implements a conservative timing model ($8 \times 8 \times 8$) and a set of LRS-metadata maintenance techniques that offer *higher* but efficient estimation of the LRS-counters (Section 4.1). Hence, LADDER is always abiding by a safety margin that ensures write reliability.

Impact of process and runtime variability. ReRAM devices from various manufacturers may have varying latency requirements (e.g., due to process technology differences). The performance benefits of LADDER typically increase for ReRAM devices with higher variations in the required RESET latency. To investigate the effectiveness of LADDER on different dynamic ranges of RESET timing, we run LADDER with a shrink of dynamic latency range in the timing table (in Section 5) by 2 ×. Notably, our results show that LADDER is able to retain 85% of the original performance advantage on average. In addition to process variations, the memory access performance can also be impacted by fluctuations in operating conditions. Particularly, prior characterization studies have shown that the threshold voltage for the resistive state of ReRAM cells can vary based on temperature [33]. To account for such runtime factors, one possible way is to provide certain latency margin in the timing table to accommodate all scenarios (JEDEC supports up to +85°C [23]). Note that such provisioning is also necessary for conventional ReRAM systems and is not unique to LADDER. To limit the performance impact due to worst-case latency settings, LADDER can utilize a few sets of timing tables, each corresponds to a certain temperature range, similar to the use of adaptive access latency in DRAM [28].

Security implications of LADDER design. LADDER utilizes in-memory content-dependent latency for writes, which can potentially raise concerns of side channels [12–15, 22, 31, 36, 54–57] where adversaries may attempt to harness the write timing to exfiltrate data stored in memory. While theoretically possible, observing exact write service latency is difficult since writes are not in the critical path. Moreover, write latency used in LADDER is only loosely dependent on the *estimated worst-case data pattern* at the page level. Differential latency observed at application user-space (if any) is bound to be too coarse-grained and non-deterministic to be leveraged for inferring useful information illegitimately. Finally, for security-wary users, lightweight system-level confinement policies can be set so that mutually-distrusting application domains do not share the same wordline group. This can prevent a malicious party from trying to exfiltrate information about the victim's data pattern in memory. Note that a complete side channel analysis in the ReRAM main memory is out of the scope of this paper.

8 RELATED WORK

There are several prior works on optimizing the write performance of crossbar-based ReRAM memories. *DSWD* [65] augments write drivers in both sides of the bitline to reduce the IR drop along bitlines. *Transpose Memory* [46] adds additional decoder circuitry and sense amplifiers in both edges of the bitlines for voltage isolation. *DAWS* [48] utilizes a path-dependent voltage biasing scheme to reduce the effective path length of sneak current. Recently, Zokaee *et al.* [68] propose a dynamic power regulation mechanism that applies higher voltage to cells suffering from greater sneak current leakage. Although such circuit-level enhancements alleviate the sneak current, these techniques do not completely eliminate IR drop and thus the RESET latency variation still exists. Leader [62] remaps frequently accessed pages to wordlines closer to the write driver to optimize the write performance. However, this approach only considers the location dependency and it is also not compatible with existing wear-leveling techniques. *Bitline profiling (BLP)* [50] utilizes a bitline content dependent write mechanism and uses additional circuitry in the memory to profile data patterns in the bitlines. Such approaches require non-trivial circuit-level support in the memory system, undermining the area and cost efficiency of ReRAM main memory. Xu et al. [52] propose the *Split-reset* write scheduling mechanism that offers two RESET speed grades where compressible data lines can benefit from shorter write latency. This technique does not take advantage of ReRAM data patterns that can lead to considerable write latency variations. Different from prior studies, we note that LADDER is the first work to enable the processor-side write optimization framework that leverages both content and location dependencies in ReRAM write operations. Our proposed LRS-metadata management scheme involves modest changes to the memory controller with minimal hardware cost and offers considerable performance advantages over state-of-the-art works.

Several works focus on enhancing the lifetime of NVM devices by reducing the effective number of bits written to the main memory (e.g., [32] and [11]). *Mellow-write* [63] extends the lifetime of ReRAM cells by selectively performing slower writes under reduced RESET voltage to ReRAM cells. Additionally, by exploiting the bank-idle times, PreSET [38] preemptively initializes bits prior to the writing of a dirty cache line to the PCM memory. Several previous studies have investigated the use of adaptive address remapping of write-intensive pages to locations that have lower RESET latency requirement [48, 51, 62]. LADDER can potentially incorporate these techniques to further improve its performance and efficacy.

9 CONCLUSION

ReRAM memories are subject to variable write latency due to the varying impact of IR drop to RESET operations. In this paper, we propose LADDER, a processor-side mechanism in the memory controller that improves the overall performance of ReRAM based system by enabling content and location-aware writes. LADDER integrates an LRS-metadata based design that keeps track of per-row data patterns completely within the memory controller. We demonstrate a basic LADDER design and propose several optimizations to further improve the performance while reducing the run-time overhead. We implement LADDER on a cycle-level simulator and evaluate its performance using 16 workloads from SPEC2006 and PARSEC. Evaluation results show that LADDER is able to achieve on average 46% performance gain compared to a baseline and 13.2% over a state-of-the-art technique. Finally, LADDER achieves 28.8% dynamic memory energy savings compared to existing architectural schemes and has less than 3% impact on device lifetime.

ACKNOWLEDGMENTS

This work is supported in part by the U.S. National Science Foundation under award numbers CNS-2008339 and CNS-1908471.

REFERENCES

[1] 2010. Intel® Core™ i7-930 Processor (8M Cache, 2.80 GHz, 4.80 GT/s Intel® QPI) Product Specifications. https://ark.intel.com/content/www/us/en/ark/products/41447/intel-core-i7-930-processor-8m-cache-2-80-ghz-4-80-gt-s-intel-qpi.html

[2] 2014. Annex K: Serial Presence Detect (SPD) for DDR3 SDRAM Modules. https://www.jedec.org/sites/default/files/docs/4_01_02_11R24.pdf

[3] 2017. JEDEC Standard: DDR4 SDRAM JESD79-4B. https://www.jedec.org/standards-documents/docs/jesd79-4a

[4] 2019. The Machine: A new kind of computer. https://www.hpl.hp.com/research/systems-research/themachine/ https://www.labs.hpe.com/memory-driven-computing.

[5] Hiroyuki Akinaga and Hisashi Shima. 2010. Resistive random access memory (ReRAM) based on metal oxides. *Proc. IEEE* 98, 12 (2010), 2237–2251.

[6] Rajeev Balasubramonian, Andrew B Kahng, Naveen Muralimanohar, Ali Shafiee, and Vaishnav Srinivas. 2017. CACTI 7: New tools for interconnect exploration in innovative off-chip memories. *ACM TACO* 14, 2 (2017), 1–25.

[7] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. 2008. The PARSEC benchmark suite: Characterization and architectural implications. In *IEEE PACT*. 72–81.

[8] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. 2011. The gem5 simulator. *ACM CAN* 39, 2 (2011), 1–7.

[9] Jalil Boukhobza, Stéphane Rubini, Renhai Chen, and Zili Shao. 2017. Emerging NVM: A survey on architectural integration and research challenges. *ACM TODAES* 23, 2 (2017), 1–32.

[10] Meng-Fan Chang, Albert Lee, Pin-Cheng Chen, Chrong Jung Lin, Ya-Chin King, Shyh-Shyuan Sheu, and Tzu-Kun Ku. 2015. Challenges and circuit techniques for energy-efficient on-chip nonvolatile memory using memristive devices. *IEEE JETCAS* 5, 2 (2015), 183–193.

[11] Sangyeun Cho and Hyunjin Lee. 2009. Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance. In *IEEE MICRO*. 347–357.

[12] Md Hafizul Islam Chowdhuryy, Rickard Ewetz, Amro Awad, and Fan Yao. 2021. Seeds of SEED:R-SAW: New Side Channels Exploiting Read Asymmetry in MLC Phase Change Memories. In *IEEE SEED*.

[13] Md Hafizul Islam Chowdhuryy, Hang Liu, and Fan Yao. 2020. BranchSpec: Information Leakage Attacks Exploiting Speculative Branch Instruction Executions. In *IEEE ICCD*.

[14] Md Hafizul Islam Chowdhuryy and Fan Yao. 2021. Leaking Secrets through Modern Branch Predictor in the Speculative World. *arXiv preprint arXiv:2107.09833* (2021).

[15] Dmitry Evtyushkin, Ryan Riley, Nael CSE Abu-Ghazaleh, ECE, and Dmitry Ponomarev. 2018. Branchscope: A new side-channel attack on directional branch predictor. *ACM SIGPLAN Notices* 53, 2 (2018), 693–707.

[16] Richard F Freitas and Winfried W Wilcke. 2008. Storage-class memory: The next storage system technology. *IBM Journal of Research and Development* 52, 4.5 (2008), 439–447.

[17] Darryl Gove. 2007. CPU2006 working set size. *ACM CAN* 35, 1 (2007), 90–96.

[18] B Govoreanu, G Kar, Y Chen, V Paraschiv, S Kubicek, A Fantini, IP Radu, L Goux, S Clima, R Degraeve, et al. 2011. 10x10nm2 Hf/HfOx Cross-point Resistive RAM with Excellent Performance, Reliability and Low-Energy Operation. In *IEEE IEDM*. 31–6.

[19] John L Henning. 2006. SPEC CPU2006 benchmark descriptions. *ACM CAN* 34, 4 (2006), 1–17.

[20] Chung-Wen Ho, A. Ruehli, and P. Brennan. 1975. The modified nodal approach to network analysis. *IEEE TCAS* 22, 6 (June 1975), 504–509.

[21] Intel. 2021. eADR: New Opportunities for Persistent Memory Applications. https://software.intel.com/content/www/us/en/develop/articles/eadr-new-opportunities-for-persistent-memory-applications.html/

[22] Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. 2015. S $ A: A shared cache attack that works across cores and defies VM sandboxing–and its application to AES. In *IEEE S&P*. 591–604.

[23] JEDEC. 2014. JEDEC Standard No. 21C: Low Power Double Data Rate (LPDDR) Non-Volatile Memory (NVM). https://www.jedec.org/sites/default/files/docs/3_06_03R18A.pdf

[24] Lei Jiang, Youtao Zhang, Bruce R Childers, and Jun Yang. 2012. FPB: Fine-grained power budgeting to improve write throughput of multi-level cell phase change memory. In *IEEE MICRO*. 1–12.

[25] Akifumi Kawahara, Ryotaro Azuma, Yuuichirou Ikeda, Ken Kawai, Yoshikazu Katoh, Yukio Hayakawa, Kiyotaka Tsuji, Shinichi Yoneda, Atsushi Himeno, Kazuhiko Shimakawa, et al. 2012. An 8 Mb multi-layered cross-point ReRAM macro with 443 MB/s write throughput. *IEEE JSSC* 48, 1 (2012), 178–185.

[26] Benjamin C Lee, Engin Ipek, Onur Mutlu, and Doug Burger. 2009. Architecting phase change memory as a scalable dram alternative. In *IEEE ISCA*. 2–13.

[27] 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.

[28] Donghyuk Lee, Yoongu Kim, Gennady Pekhimenko, Samira Khan, Vivek Seshadri, Kevin Chang, and Onur Mutlu. 2015. Adaptive-latency DRAM: Optimizing DRAM timing for the common-case. In *IEEE HPCA*. 489–501.

[29] Charles Lefurgy, Karthick Rajamani, Freeman Rawson, Wes Felter, Michael Kistler, and Tom W Keller. 2003. Energy management for commercial servers. *IEEE Computer* 36, 12 (2003), 39–48.

[30] Jianhua Li, Chun Jason Xue, and Yinlong Xu. 2011. STT-RAM based energy-efficiency hybrid cache for CMPs. In *IEEE VLSI-SoC*. 31–36.

[31] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. 2015. Last-level cache side-channel attacks are practical. In *IEEE S&P*. 605–622.

[32] Rakan Maddah, Seyed Mohammad Seyedzadeh, and Rami Melhem. 2015. CAFO: Cost aware flip optimization for asymmetric memories. In *IEEE HPCA*. 320–330.

[33] Anas Mazady and Mehdi Anwar. 2014. Memristor: part II–DC, transient, and RF analysis. *IEEE T-ED* 61, 4 (2014), 1062–1070.

[34] Dimin Niu, Cong Xu, Naveen Muralimanohar, Norman P Jouppi, and Yuan Xie. 2012. Design trade-offs for high density cross-point resistive memory. In *IEEE ISLPED*. 209–214.

[35] 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 *IEEE ICCAD*. 17–23.

[36] Dag Arne Osvik, Adi Shamir, and Eran Tromer. 2006. Cache attacks and countermeasures: the case of AES. In *Springer CT-RSA*. 1–20.

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

[38] Moinuddin K Qureshi, Michele M Franceschini, Ashish Jagmohan, and Luis A Lastras. 2012. PreSET: Improving performance of phase change memories by exploiting asymmetry in write times. *ACM CAN* 40, 3 (2012), 380–391.

[39] 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 *IEEE MICRO*. 14–23.

[40] Moinuddin K Qureshi and Yale N Patt. 2006. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In *IEEE MICRO*. 423–432.

[41] Moinuddin K Qureshi, Vijayalakshmi Srinivasan, and Jude A Rivers. 2009. Scalable high performance main memory system using phase-change memory technology. In *IEEE ISCA*. 24–33.

[42] Simone Raoux, Geoffrey W Burr, Matthew J Breitwisch, Charles T Rettner, Y-C Chen, Robert M Shelby, Martin Salinga, Daniel Krebs, S-H Chen, H-L Lung, et al. 2008. Phase-change random access memory: A scalable technology. *IBM Journal of Research and Development* 52, 4.5 (2008), 465–479.

[43] 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. In *IEEE ISCA*. 383–394.

[44] J. E. Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W. R. Davis, P. D. Franzon, M. Bucher, S. Basavarajaiah, J. Oh, and R. Jenkal. 2007. FreePDK: An Open-Source Variation-Aware Design Kit. In *IEEE MSE*. 173–174.

[45] Dmitri B Strukov, Gregory S Snider, Duncan R Stewart, and R Stanley Williams. 2008. The missing memristor found. *nature* 453, 7191 (2008), 80–83.

[46] Nishil Talati, Saransh Gupta, Pravin Mane, and Shahar Kvatinsky. 2016. Logic design within memristive memories using memristor-aided loGIC (MAGIC). *IEEE TNANO* 15, 4 (2016), 635–650.

[47] Ronald Tetzlaff. 2013. *Memristors and memristive systems*. Springer.

[48] Chengning Wang, Dan Feng, Jingning Liu, Wei Tong, Bing Wu, and Yang Zhang. 2017. Daws: Exploiting crossbar characteristics for improving write performance of high density resistive memory. In *IEEE ICCD*. 281–288.

[49] Wen Wen, Lei Zhao, Youtao Zhang, and Jun Yang. 2017. Speeding up crossbar resistive memory by exploiting in-memory data patterns. In *IEEE ICCAD*. 261–267.

[50] Wen Wen, Lei Zhao, Youtao Zhang, and Jun Yang. 2019. Exploiting In-memory Data Patterns for Performance Improvement on Crossbar Resistive Memory. *IEEE TCAD* (2019).

[51] Bing Wu, Dan Feng, Wei Tong, Jingning Liu, Shuai Li, Mingshun Yang, Chengning Wang, and Yang Zhang. 2018. Aliens: A novel hybrid architecture for resistive random-access memory. In *IEEE ICCAD*. 1–8.

[52] 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 *IEEE HPCA*. 476–488.

[53] 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 *IEEE DAC*. 1–6.

[54] Mengjia Yan, Read Sprabery, Bhargava Gopireddy, Christopher Fletcher, Roy Campbell, and Josep Torrellas. 2019. Attack directories, not caches: Side channel attacks in a non-inclusive world. In *IEEE S&P*. 888–904.

[55] Fan Yao, Milos Doroslovacki, and Guru Venkataramani. 2018. Are coherence protocol states vulnerable to information leakage?. In *IEEE HPCA*. 168–179.

[56] Fan Yao, Hongyu Fang, Miloš Doroslovački, and Guru Venkataramani. 2019. Leveraging cache management hardware for practical defense against cache timing channel attacks. *IEEE Micro* 39, 4 (2019), 8–16.
[57] Fan Yao, Guru Venkataramani, and Miloš Doroslovački. 2017. Covert timing channels exploiting non-uniform memory access based architectures. In *ACM GLSVLSI*. 155–160.
[58] Mao Ye, Clayton Hughes, and Amro Awad. 2018. Osiris: A low-cost mechanism to enable restoration of secure non-volatile memories. In *IEEE MICRO*. 403–415.
[59] Vinson Young, Prashant J. Nair, and Moinuddin K. Qureshi. 2015. DEUCE: Write-Efficient Encryption for Non-Volatile Memories. In *ASPLOS*. 33–44.
[60] Shimeng Yu and Pai-Yu Chen. 2016. Emerging memory technologies: Recent trends and prospects. *IEEE SSC-M* 8, 2 (2016), 43–56.
[61] Shimeng Yu and H-S Philip Wong. 2010. A phenomenological model for the reset mechanism of metal oxide RRAM. *IEEE EDL* 31, 12 (2010), 1455–1457.
[62] Hang Zhang, Nong Xiao, Fang Liu, and Zhiguang Chen. 2016. Leader: Accelerating reram-based main memory by leveraging access latency discrepancy in crossbar arrays. In *IEEE DATE*. 756–761.
[63] 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 *IEEE ISCA*. 519–531.
[64] Sen Zhang, Shibing Long, Weihua Guan, Qi Liu, Qin Wang, and Ming Liu. 2009. Resistive switching characteristics of MnOx-based ReRAM. *Journal of Physics D: Applied Physics* 42, 5 (2009), 055112.
[65] Yang Zhang, Dan Feng, Jingning Liu, Wei Tong, Bing Wu, and Caihua Fang. 2017. A novel reram-based main memory structure for optimizing access latency and reliability. In *IEEE DAC*. 1–6.
[66] Lei Zhao, Lei Jiang, Youtao Zhang, Nong Xiao, and Jun Yang. 2017. Constructing fast and energy efficient 1tnr based reram crossbar memory. In *IEEE ISQED*. 58–64.
[67] Ping Zhou, Bo Zhao, Jun Yang, and Youtao Zhang. 2009. A Durable and Energy Efficient Main Memory Using Phase Change Memory Technology. In *IEEE ISCA*. 14–23.
[68] Farzaneh Zokaee and Lei Jiang. 2020. Mitigating Voltage Drop in Resistive Memories by Dynamic RESET Voltage Regulation and Partition RESET. In *IEEE HPCA*. 275–286.
[69] Kazi Abu Zubair and Amro Awad. 2019. Anubis: ultra-low overhead and recovery time for secure non-volatile memories. In *IEEE MICRO*. 157–168.
[70] Pengfei Zuo, Yu Hua, Ming Zhao, Wen Zhou, and Yuncheng Guo. 2018. Improving the performance and endurance of encrypted non-volatile main memory through deduplicating writes. In *IEEE MICRO*. 442–454.