Leader: Accelerating ReRAM-based Main Memory by Leveraging Access Latency Discrepancy in Crossbar Arrays

Hang Zhang, Nong Xiao, Fang Liu, Zhiguang Chen
State Key Laboratory of High Performance Computing, College of Computer
National University of Defense Technology, Changsha, Hunan 410073, China
Email: {hangzhang, nongxiao, liufang, chenzhiguang}@nudt.edu.cn

***Abstract*—Emerging Resistive Memory (ReRAM) technology is a promising candidate as the replacement to DRAM due to its low leakage power consumption, good scalability, and high density. By employing crossbar structures, the density of ReRAM can be further improved for capacity benefits. However, such structure also causes an IR drop issue due to wire resistance and sneak currents, which lead to an access latency discrepancy in ReRAM memory banks. Existing designs conservatively utilize the worst-case latency of ReRAM arrays, and thus fail to explore the potential of the fast access speed of ReRAM, resulting in sub-optimal performance.**

In this work, we present an asymmetric ReRAM memory design, which separates a crossbar array into multiple logical regions according to their access latency, and further groups logical regions across different crossbars into virtual regions. Based on the observation of access hotspots inside memory banks, we design a table structure to remap memory requests to different virtual regions with non-uniform access latency, so as to match these access hotspots with the underlying asymmetric bank design. We then introduce both static mapping and dynamic mapping schemes to prioritize memory requests from critical applications to the fast regions for better performance.

Experimental results show that our design can improve the 4-core system performance by 13.3% and reduce the memory latency by 21.6% on average for a ReRAM-based memory system across memory intensive applications.

I. Introduction

As technology scales, DRAM faces great challenges, requiring frequent refreshing operations for high-leakage cells, having longer write recovery time, and being difficulty to build high-density arrays [1]. Consequently, emerging non-volatile memory (NVM) technologies are explored in possible alternatives to DRAM, because of their promising characteristics, including high density, near-zero power leakage, good scalability, and non-volatility [2][3]. Among these NVM candidates, ReRAM stands out with higher density, better scalability, longer endurance, and relatively lower write energy [4][5][6][7].

Thanks to their nonlinearity feature, ReRAM cells can be constructed into a crossbar structure without access transistors to achieve very high density. In addition, multiple layers of crossbars can be stacked together via 3D integration technologies for even higher density. The recent industrial prototypes from SanDisk [8] and Micron [9] have demonstrated the possibility of incorporating ReRAM into current computer systems as main memory or massive storages. Intel and Micron have also announced a novel 3D XPoint non-volatile memory product that is built with a new non-volatile memory technology in 3D crossbars [10].

Although the crossbar structure enables high-density ReRAM arrays, it also entails non-trivial complexities and challenges when designing read/write mechanisms that are energy-efficient and reliable. The magnitude of IR drops along metal wires increases dramatically as the size of arrays becomes larger, and may eventually lead to unreliable write operations and write failures. Besides, it is exacerbated by the widely adopted *half-biased* write scheme, resulting from escalating currents flowing over crossbars by applying a biased half voltage on the unselected cells. As technology scales, this IR drop issue will be further aggravated due to increased wire resistance, and may finally limit the size of crossbar arrays, imposing constraints on ReRAM density. Furthermore, this IR drop issue accounts for a great access latency discrepancy among memory rows inside a crossbar, because the switching time of a ReRAM cell is exponentially inverse to the write voltage applied to the cell. Previous designs with conservative worst-case latency access of a ReRAM array can result in pessimistic performance of using ReRAM.

To address the challenges caused by IR drops, we propose a novel asymmetric ReRAM memory bank organization, ***Leader***, to ***LE**verage the **A**ccess latency **D**iscrepancy of R**eR**AM arrays. By identifying disparate write latencies among different rows inside a crossbar ReRAM array, we group contiguous rows together as a logical region with regard to their access latency. As a result, A memory bank is divided into a series of logical regions, each of which can be accessed with different write latencies. We then propose a virtual region-grouping scheme to provide the flexibility of remapping memory requests to different regions according to various performance requirements. By adopting our proposed static and dynamic mapping schemes, the asymmetric ReRAM design has demonstrated a substantial memory latency reduction and performance improvement across a wide variety of applications via experiments.

We make the contributions in this work as follows:

- We study the IR drop effect via a detailed circuit-level model to demonstrate the access latency discrepancy.
- We analyze the hotspots upon row accesses inside memory banks. Based on this observation, we present a virtual region-grouping design providing the flexibility of remapping memory requests.
- We propose two virtual region mapping schemes, static and dynamic mapping to adaptively remap row addresses according to the access pattern of applications for better performance.
- We quantitatively evaluate our proposed design and show a performance improvement of 13.3% for memory intensity applications on average.

To the best of our knowledge, this is the first work to explore the non-uniformity of row access latencies inside a crossbar

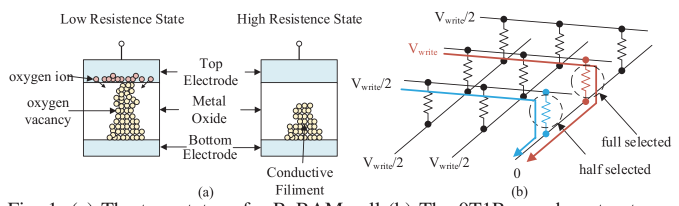
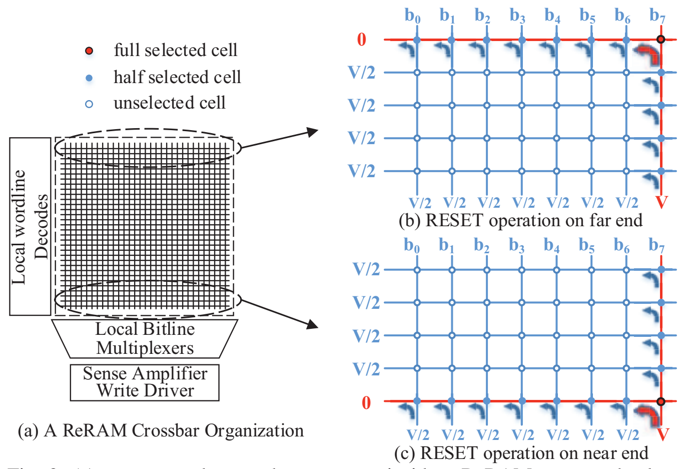


Fig. 1: (a) The two states of a ReRAM cell (b) The 0T1R crossbar structure



full selected cell
half selected cell
unselected cell
Local wordline Decodes
Local Bitline Multiplexers
Sense Amplifier Write Driver
(a) A ReRAM Crossbar Organization
b_0 b_1 b_2 b_3 b_4 b_5 b_6 b_7
0
V/2
V
(b) RESET operation on far end
(c) RESET operation on near end

Fig. 2: (a) represents the crossbar structure inside a ReRAM memory bank. (b)(c) demonstrates the IR drops along the far end row and the near end row under RESET operation, respectively.

of ReRAM-based memory. We argue that, for the ReRAM based main memory system, the characteristics of discrepancy should be exposed to the memory controller to unleash the potential of performance-improving, which is different from the conventional DRAM-based memory system design.

II. Preliminary

A ReRAM cell consists of three layers. A metal-oxide resistive switch is sandwiched between a top metal electrode and a bottom metal electrode. The conceptual ReRAM cell structure is shown in Fig. 1(a). There are various metal oxide materials that have the feasibility to build the resistive switch layer, such as TiOx [11]. They usually result in cells with different characteristics. The switch layer has various states depending on whether its internal conductive filament is formed or ruptured, leading to the ReRAM cell with various resistances.

A ReRAM cell makes use of its two distinguished resistance states to represent logic values. The high resistance state (HRS) denotes logic "0", while the low resistance state (LRS) denotes logic "1". To change the resistance of a ReRAM cell, a voltage with particular magnitude and polarity is applied to the cell, which is referred as either SET or RESET operation. The SET operation is used to change the ReRAM cell state from HRS to LRS, while the RESET operation changes ReRAM cell state from LRS to HRS.

III. Motivation

In this section, we describe the IR drop issue on crossbar structures, and analyze the latency discrepancy inside the crossbar structure due to the IR drop issue.

A. IR Drop Issue

Crossbar structures suffer from the IR drop issue, which is caused by wire resistance along bitlines and sneak currents.

The wire resistance is dramatically increased as the technology scales. To analyze the wire resistance, the row that is physically near to the write driver is referred as the near end row, while the row that is physically far from the write driver is referred as the far end row. As shown in Fig. 2, to RESET a cell in a far end row, it requires currents to flow through a longer bitline than to RESET a cell in a near end row. Therefore, this difference on the bitline length results in a variance of the IR drops among different rows when performing RESET operations.

TABLE I: Parameters in the Crossbar Array Model

Metric	Description	Value
A	Mat size: A wordlines×A bitlines M	1024
n	Number of bits to read/write	1
I_{on}	Cell current of a LRS ReRAM during RESET	$15uA$
R_{wire}	Wire resistance between adjacent cells	20Ω
K_r	Nonlinearity of the selector	3000
V_W	Full selected voltage during write	$3.2V$
V_R	Read voltage	$1.6V$

The sneak current is a result of the *half-biased write* scheme that is originally designed to suppress the sneak path issue of crossbar structures [4]. In order to SET/RESET a ReRAM cell inside a crossbar, particular voltages are applied to the selected wordline and bitline. To RESET the target ReRAM cell in crossbars, the selected wordline has to be applied is grounded at "0", while the selected bitline is applied with a full voltage swing V_{RESET} at the same time. As shown in Fig. 1b, the cells that are applied with full voltage are referred as *fully-selected cells*, while the other cells along the selected wordline or bitline(s) are referred as *half-selected cells*. Since there is no access transistor to isolate ReRAM cells in crossbars any more, the *half-selected cells* also experience full voltage, which results in current flowing through *half-selected cells*. These currents are usually referred as sneak currents. The existence of such sneak currents also aggravates the IR drop issue, since more unselected cells are attached to the bitline when performing writes on far end rows.

Unfortunately, the existence of the IR drop issue leads to various applied voltages on top of ReRAM cells when performing SET/RESET operations. Such various voltage drops may further result in unreliable write operations and performance divergence,

B. RESET Latency Discrepancy

The latency of write operations is composed of the SET latency and the RESET latency. Because the SET operation is usually completed relatively faster, the slower RESET operation becomes the performance bottleneck [5].

One critical characteristic of an ReRAM cell is that its switching time is exponentially inverse to the voltage drop applied on the cell [11]. The RESET latency of a ReRAM cell is calculated by the following equation, $t \times e^{kV_d} = C$, where t denotes the switching latency and V_d denotes the voltage drop across the cell. Both k and C are fitting constants from experiments. Since rows at different locations of a crossbar array experience different applied voltages as we mentioned, this characteristic triggers RESET latency variances among rows with different locations inside the same ReRAM array.

To quantitatively characterize the relationship of the row location, the voltage drops, and the RESET latency, we build a detailed HSPICE circuit model to analyze crossbar arrays under the 14nm technology. In this work, we choose the single-bit programming scheme rather than the multiple-bits programing scheme, since the multiple-bits programing scheme is more susceptible to the IR drops issue and has limited scalability on large-size crossbars [5]. The key parameters are illustrated in Table I, which is derived from the previous work [5] and ITRS [12]. The RESET latency of a row is measured as the latency of the worst-case cell that is furthest from the row decoder. Fig. 3 shows the relationship of voltage

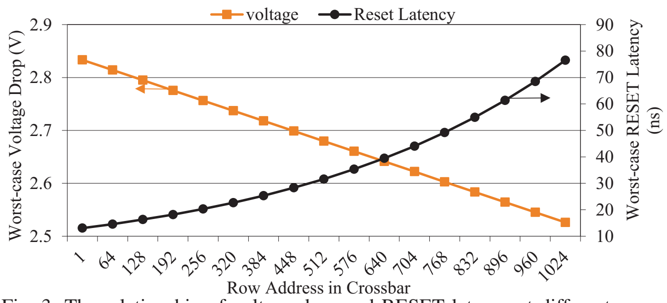


Fig. 3: The relationship of voltage drop and RESET latency at different row addresses inside a crossbar array.

drops and the RESET latency for different row addresses inside a crossbar. The RESET latency is evaluated with a granularity of 64 rows, and the index of row addresses starts from the row that is nearest to the write driver. It is clear to see that there is a great RESET latency discrepancy between the near end rows and far end rows. The row #1 only has a 15ns RESET latency, while the row #1024 has a 78ns RESET latency.

The farther a row is from the write driver, the larger IR drop that row experiences. Therefore, the switching latency for RESET of the far end rows is much longer than that of the near end rows, thereby resulting in an increasing RESET latency discrepancy inside of ReRAM memory banks. However, in previous designs, the write access latency to a ReRAM memory bank is always conservatively set to be the worst-case latency of the most far end row. Although this setting guarantees the correctness of write operations, it loses opportunities for potential performance improvement.

IV. Asymmetric ReRAM Bank Design

In this section, we first analyze write traffic patterns inside memory banks. We then propose a novel asymmetric ReRAM design, which partitions a ReRAM memory bank into logical regions with different write access latencies, and makes the memory controller aware of the variances on access latencies.

A. *Skewed Memory Access Pattern*

To understand write traffic distributions of different applications, we study memory access patterns for a wide range of benchmarks from SPEC CPU 2006 [13]. We model a DDRx-compatible 2Gb x8 ReRAM chip with 8 banks in our work. We ran all the benchmarks at various regions of interests for 1 billion instructions, and then analyzed access patterns at the granularity of a mat. The detailed experimental configuration is described in Section V. A mat is the basic building block in the memory array design, and each memory mat contains a 1024x1024 crossbar. A ReRAM memory bank is composed of 256 mats in total. We first take a look at the access pattern of requested memory addresses from the memory controller perspective. The case of cactusADM benchmark is taken as an example, and the statistic is shown in Fig. 4. The requested memory addresses are evenly distributed to the address space, and the maximum access counts does not even exceed 25. However, if we inspect the region access frequency in Fig. 5, it is interesting to see that the accesses to each memory region are not evenly distributed. Every 64 rows are evaluated together as a region. Only a small portion of regions account for most of the memory traffic, which results in the access hotspots. In addition, the difference in access frequency for different regions varies at a wide range. Similar results for other benchmarks have also been observed but not included here due to the space limitation.

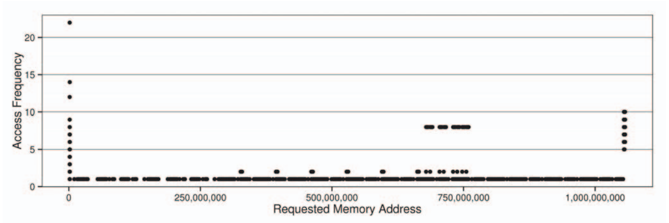


Fig. 4: Acess frequency on requested memory address for cactusADM benchmark.

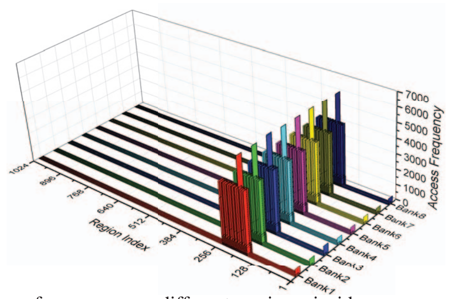


Fig. 5: Acess frequency on different regions inside a memory bank for cactusADM benchmark.

The existence of access hotspots on a small number of regions inside a mat might be counter-intuitive. It is expected to have uniformly distributed memory accesses to each region, since the requested memory addresses are evenly distributed. However, if we take a closer look at the address-mapping scheme adopted by memory systems, it is not hard to understand the reason behind this access non-uniformity. Basically, the access hotspots result from a sharing of row addresses among these memory requests, since memory systems usually designate the most significant bits of addresses as the row address. Memory systems usually employ hierarchal structures to organize DRAM cells, such as channel, rank, bank, subarray, and mat. Each level of these structures has its own independence to operate without interfering the others, which is referred as the structural parallelism.

To obtain the best parallelism, memory systems adopt a particular mapping scheme to map physical memory addresses onto these memory organizations. It is usually referred as the address mapping scheme. Here we use r, ra, b, ch and c to represent row, rank, bank, channel and column, respectively. For instance, memory systems prefer to use this r:ra:b:ch:c configuration as the optimal address mapping scheme to minimize bank conflicts for an open-page based system [14]. Therefore, a part of the most significant bits of a memory address are designated to address rows inside a bank. Consequently, contiguous physical memory pages that share common significant address bits are mapped to the same row address, thereby leading to the access hotspots inside memory banks.

Moreover, modern operating systems also tend to allocate contiguous page frames to satisfy a memory allocation request from an application, owing to the simplicity of memory allocation managements, such as the buddy-allocation scheme adopted in Linux systems [15]. As a result, accesses to these contiguous page frames from an application are very likely to fall into the same regions of different banks, thus increasing the possibility of raising access hotspots inside memory banks.

B. *Partition ReRAM Memory Arrays into Regions*

In order to match the access hotspots of memory traffic with the inherent characteristic of various row access latencies on

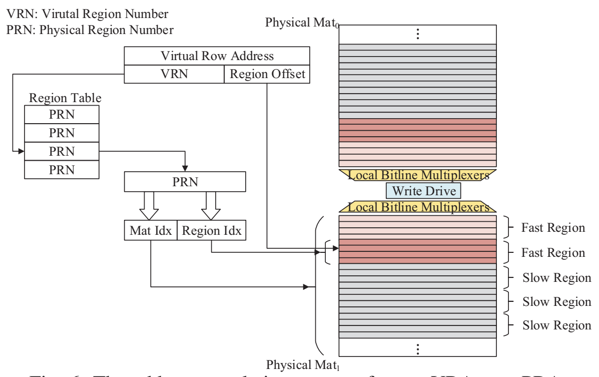


Fig. 6: The address translation process from a VRA to a PRA.

Algorithm 1: Dynamic-Mapping

```
Initialize Pysical Region Scores = $\{S_1, ..., S_N\}$
Initialize Write Scores $Scores = \{WS_1, ..., WS_N\}$
Initialize Read Scores $Scores = \{RS_1, ..., RS_N\}$
Function Migration(Scores)
    for $i := 1$ to $RegionsNumber$ do
        $S_i = \alpha \times WS_i + \beta \times RS_i$
    end
    if $(S_{max} - S_{min}) \geq$ Threshold then
        Swap ($VRN_{max}$, $VRN_{min}$)
        Update (RegionTable)
        Half (Scores)
    end
Function WriteDrain(Scores)
    for $r$ in Read/Write Queue do
        if $r.type$ ==WRITE then
            $WS_{r.RegionNum}$++
        else
            $RS_{r.RegionNum}$++
        end
    end
```

ReRAM memory arrays, we propose a virtual region scheme to provide the flexibility to remap row addresses inside memory banks. We group multiple contiguous rows together into a logical region. Therefore, a ReRAM memory bank consists of several logical regions. These regions inside a memory bank can have different write latencies.

The virtual region mapping design includes two abstractions, *Physical Region* (PR) and *Virtual Region* (VR). A *Physical Region* (PR) is composed of 1∼1024 row(s). For dicussion simplicity, a region is composed of 64 rows in this work. Therefore, there are multiple PRs inside a mat. The total number of PRs inside a memory bank is calculated as the number of PRs per mat multiplied by the number of mats in a bank. Though it is technically possible to assign different latencies to each PR, we adopt two types of PRs in this work. They are *Fast Physical Region* (FPR) and *Slow Physical Region* (SPR). FPRs have the lower write latency of the near end rows, while SPRs have the write latency of far end rows as worst-case. Each PR has a unique *Physical Region Number* (PRN). The row address used in conventional address mapping schemes now becomes the virtual row address (VRA). A series of virtual row addresses compose a Virtual Region (VR). Each VR also has a unique *Virtual Region Number* (VRN). A VR has the same number of rows as one PR. A *Region Table* (RT) is devised to record the mapping from VRNs to PRNs for each bank. The RT has a role similar to *Page Table* for the virtual memory system of operating systems.

The address translation process from a virtual row address (VRA) to a physical row address (PRA) is illustrated in Fig. 7. A VRA consists of two parts, *Virtual Region Number* (VRN) and *Region Offset* (RO). The VRN part is used to index the *Region Table* to fetch PRN, while the RO part is used to specify which physical row inside the PN. If the acquired PRN belongs to FPR, the corresponding VPR can be accessed with low write latency; if the acquired PRN belongs to SPRs, the corresponding VPR is accessed with the normal access latency.

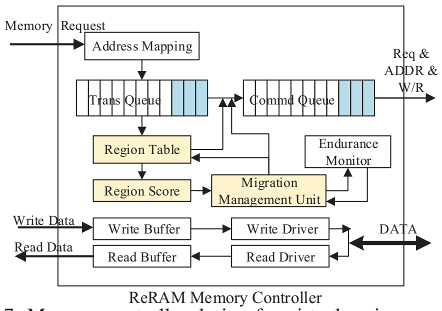


Fig. 7: Memory controller design for virtual region mapping

In order to determine whether a PRN is a FPR or a SPR, the PRN is further broken down into two parts, the *Mat Index* (MI) and the *Region Index* (RI). The MI is used to index which mat contains the PRN, and the number of bits of MI field is equal to the logarithm of the number of total mats inside a bank. The RI is used to index the PR inside each mat, and the length of the RI field is equal to the logarithm of the number of how many physical rows a PR contains. Thus, we can simply use the MI combined with the significant bits of the RI to determine the fast regions. By using this VR design, the scattered fast regions from different mats can be gathered to serve memory requests from critical applications.

C. Region Remapping and Migration

We propose two schemes to manage the mapping from VRN to PRN for remapping requests for performance improvement. They are static mapping and dynamic mapping.

Static Mapping Either compiler assisted or profiling based methods can be leveraged to acquire the access hotspots information with regard to the address mapping scheme adopted by the memory controller. Such information is used to setup the static-mapping by updating the *RegionTable* via system calls of the operating system. In this work, we collect the profiling information from the compiler, and then setup region mapping according to the program behaviors.

Dynamic Mapping We propose a dynamic mapping that leverages a run-time adaptive region re-mapping technique to achieve performance benefits. Based on the observation of the access hotspots of applications, we prioritize the frequently accessed virtual regions that are previously mapped to slow physical regions, and re-map them to fast physical regions. We adopt *Epoch* as a time window to decide when it needs to trigger remapping and migrating. A *Read/Write Region Score* is assigned to each virtual region for evaluating its hotness. This dynamic mapping is shown in Algorithm 1. Each time there is a need for the memory controller to perform the write queue drain, both *Write Score* and *Read Score* are updated according to the *WriteDrain* function. At the end of each *Epoch*, the *Migration* function is triggered to decide whether it needs to swap the region with the highest score and the region with the lowest score. The algorithm is based on two principles. First, the write intensive regions will gradually accumulate to high scores and will be remapped to fast regions later. Second, the number of read queuing is also taken into consideration, since the write requests that are postponing read requests tend to have higher impacts on performance. The parameter of α determines the weight of the write score and β determines the weight of the read score. Based on the

TABLE II: Simulation Hardware Configuration

CPU Cores	1/4-cores, 192-entry reorder buffer 3.0 GHz, out-of-order, 8 issue width
CPU L1 Cache	private, 32KB SRAM icache&dcache 4-way assoc, 2-cycle latency
CPU L2 Cache	shared, 4MB SRAM, 8-way assoc 64B cache line, 20-cycle latency
Main Memory	8GB DRAM/ReRAM, DDR3-1333 2 channels, 2 rank/channel, 8 bank/rank 64K rows, 4K columns, close-page mode
ReRAM Timing (ns)	tRCD(18), tCL(15), tCWD(13), tFAW(30) tWTR(7.5), tWR_{fast} (26), tWR_{slow}(86)

TABLE III: Benchmark Characteristics of SPEC CPU 2006 and STREAM

Category	#Benchmark (MPKI)
High	[1]stream(46.24), [2]lbm(25.04), [3]bzip2(17.63) [4]leslie3d(18.92), [5]bwaves(17.63), [6]sjeng(16.82), [7]zeusmp(11.36)
Medium	[8]cactusADM(7.49), [9]wrf(4.31), [10]astar(3.28) [11]perlbench(3), [12]gobmk(3.2), [13]libquantum
Low	[14]soplex(0.72), [15]mcf(0.79),[16]gromacs(0.4) [17]omnetpp(0.29), [18]xalancbmk(0.23), [19]hmmer(0.22)
MixH	mix1(1,6,2,4), mix2(3,5,6,2), mix3(2,4,1,5), mix4(7,1,6,2)
MixM	mix1(13,9,10,12),mix2(8,9,10,11),mix3(9,10,11,13),mix4(8,13,12,10)
MixL	mix1(15,18,16,17),mix2(14,16,17,18),mix3(19,15,16,14),mix4(15,16,18,19)

experimental results, we choose the configuration $\alpha = 0.5$ and $\beta = 0.5$ for the best performance. We set the length of *Epoch* to be 1 million cycles based on experiment results. Note that we just demonstrate a simple example of the address mapping scheme in this work. Our virtual region design is able to work with other mapping and migration schemes, such as previous work [16].

D. Endurance Issue

Mapping write intensive requests into fast regions may lead to endurance issues, since ReRAM cells have limited number of switching. However, this could not be a serious problem because of two reasons. First, some prototypes report ReRAM cell has comparatively higher endurance ($>10^{12}$) than that of PCM [17]. As the fabrication technology becomes mature, endurance will not be a big issue. Second, the static mapping and dynamic mapping can be considered as a performance boosting approach to satisfy critical performance requirement demanded by users. It can be turned on to favor performance benefits, while it can also be turned off to work with other wear-leveling techniques that have been devised to cope with the endurance issue of PCM, such as Row shifting [18], Start-gap [19] and Security-refresh [20].

V. Experimental Evaluation

In this section, we present evaluation results through various experiments under single-core and multi-core architectures.

A. Experimental Methodology

To evaluate our design, we use gem5 [21] simulator as the simulation framework. The detailed configuration of a simulator is illustrated in Table II. We incorporate NVMain simulator [22] into gem5 to simulate multiple channel ReRAM-based memory system. The ReRAM related memory timing parameters is derived from previous work [6].

We select benchmarks from SPEC CPU2006 [13] with reference input and STREAM with all functions [23]. We classify the benchmarks into three categories, according to the metric of misses per thousands instruction (MPKI) of each benchmark. For multi-core simulation, we randomly select four benchmarks in each category for multi-programmed simulation under 4-cores architecture. The weighted IPC [24] is adopted to evaluate the performance.

$$weighted\ speedup = \sum_{i=1}^{n}(\frac{IPC_i^{shared}}{IPC_i^{alone}})$$

Table III lists the benchmark characteristics. We run all the benchmarks for 100 million instructions to warmup caches and then run 400 million instructions for experiments with our proposed techniques.

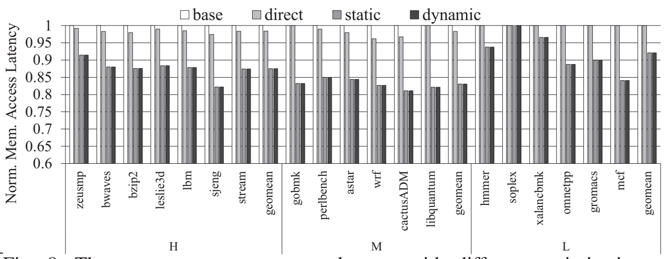


Fig. 8: The average memory access latency with different optimizations normalized to baseline.

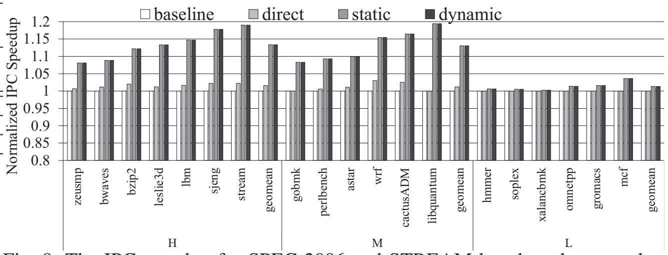


Fig. 9: The IPC speedup for SPEC 2006 and STREAM benchmarks normalized to baseline.

We choose the DSGB (double-sided ground biasing) design [5] as the aggressive baseline to show the effectiveness of our design. We compare the following configurations in our simulation. *baseline*: The ReRAM memory design with the write access latency set as worst case under DSGB. *direct*: Directly apply fast and slow regions design. *static*: Equipped with static mapping scheme to map virtual regions. *dynamic*: Equipped with dynamic mapping scheme to map virtual regions.

B. Simulation Results

Fig. 8 shows the average memory access latency for different design configurations, with results normalized to the baseline. It is clear that direct scheme only results in marginal memory latency reduction due to the under-utilized fast regions, whereas both static-mapping and dynamic-mapping can effectively reduce the average memory access latency, resulting from the decreased read queuing delay. Both the static mapping and dynamic mapping achieve promising optimized results with geometric mean 12.6% and 16.9% latency reduction for High MPKI (H) benchmarks and Medium MPKI (M) benchmarks, respectively. This attributes to the optimized region mapping by making full use of the fast regions that scatter inside memory banks and prioritizing the critical applications.

Fig. 9 shows the system speedup for different configurations normalized to baseline. Thanks to the memory access latency reduction, our static-mapping and dynamic-mapping design achieves 13.4% and 13.0% IPC improvements over the baseline for H benchmarks and M benchmarks, respectively.

Achieving this good performance improvement through our scheme stems from the better utilization the scattered fast regions in memory banks when the memory footprint of the applications is smaller than total capacity of memory. Our scheme can be leveraged to prioritize the critical applications, whose priority is defined by users, for higher performance. When the memory footprints of various applications increase, the hotpots resulting from the address mapping scheme may be obviated, but the memory traffic may remain unbalanced [16].

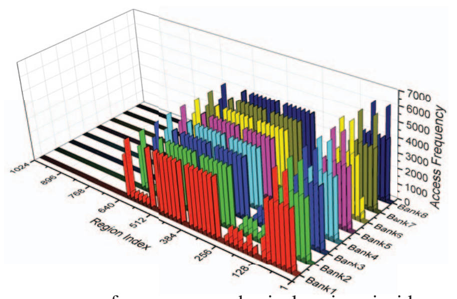


Fig. 10: Memory access frequency on physical regions inside memory banks for cactusADM benchmark after adopting static-mapping.

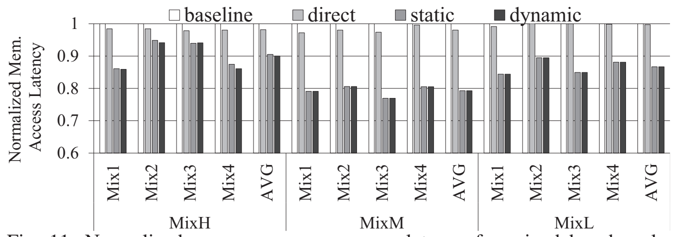


Fig. 11: Normalized average memory access latency for mixed benchmark applied to baseline.

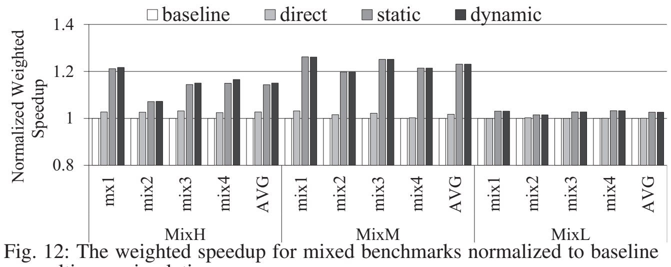


Fig. 12: The weighted speedup for mixed benchmarks normalized to baseline on multi-core simulation .

For this scenario, our proposed virtual region scheme can work together with these data migration schemes [16] for better performance.

The memory accesses inside memory banks for cactusADM benchmark after adopting static-mapping are shown in Fig. 10. Our scheme effectively alleviates access hotspots inside memory banks by scattering requests across more regions. The key for the system performance improvement is that more fast regions are utilized than the design without optimization.

For multi-programmed simulations, the normalized memory access latency comparison for mixed benchmarks is shown in Fig. 11. Our static-mapping and dynamic mapping reduce the memory access latency by geomean 21.6% and 20.7% for H benchmarks and L benchmarks, respectively. Fig. 12 demonstrates the normalized weighted speedup for mixed benchmarks. This static-mapping and dynamic-mapping can improve system performance with normalized weighted speedup at geomean 13.3% and 12.3% for H and M benchmarks, respectively.

Hardware Overhead: The hardware overhead comes with *Region Table*. To keep flexibility, the number of rows inside a PR is configurable, while that also decides how many entries a RT has. The size of RT finally results in different hardware overhead. A ReRAM bank in our design contains 64K rows, and if each PR contains only 64 rows, there are 1024 PRs in total for a bank. Thus it requires hardware overhead about $log_2(1024)\times1024$ bits. We use a 11-bit counter to store one entry of the Read/Write Score, and there are total 22×1024 bits. These structures contribute a 4KB storage requirement per bank. Note that all the chips inside a rank that is composed of 8 chips can share the same *Region Table*. Therefore, for a 2GB memory rank, it only incurs 64KB storage overhead in total for memory controller, which is negligible.

VI. Conclusion

In this work, we propose a novel asymmetric ReRAM organization to divide up memory array into regions according to access latency characteristics of rows. Based on the observation of memory access hotspots inside memory bank across various benchmarks, we further propose virtual region design to develop flexible region mapping scheme for grouping fast regions together. We also present two mapping schemes to match the access pattern with our asymmetric ReRAM bank organization. The experimental results show that our design can effectively achieve performance improvement and memory latency reduction.

VII. Acknowledgments

We are grateful to our anonymous reviewers for their suggestions to improve this paper. This work is supported by the National Natural Science Foundation of China under Grant Nos. 61433019, 61332003, 61170288, 61402503, 61502514, 61120106005 and 61303073, U1435217, and the National High Technology Research and Development 863 Program of China under Grant No. 2013AA013201 and 2015AA015306.

References

[1] U. Kang, H. Yu, C. Park, H. Zheng, J. Halbert, K. Bains *et al.*, "Co-Architecting Controllers and DRAM to Enhance DRAM Process Scaling," in *Memory Forum*, 2014.

[2] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable dram alternative," in *ISCA*, 2009.

[3] Z. Sun, X. Bi, H. H. Li, W.-F. Wong, Z.-L. Ong, X. Zhu *et al.*, "Multi retention level STT-RAM cache designs with a dynamic refresh scheme," in *MICRO*, 2011.

[4] C. Xu, X. Dong, N. P. Jouppi, and Y. Xie, "Design implications of memristor-based RRAM cross-point structures," in *DATE*, 2011.

[5] C. Xu, D. Niu, N. Muralimanohar, R. Balasubramonian, T. Zhang, S. Yu *et al.*, "Overcoming the challenges of crossbar resistive memory architectures," in *HPCA*, 2015.

[6] C. Xu, P.-y. Chen, D. Niu, Y. Zheng, S. Yu, and Y. Xie, "Architecting 3D Vertical Resistive Memory for Next-Generation Storage Systems," in *ICCAD*, 2014.

[7] C. Xu, D. Niu, N. Muralimanohar, N. P. Jouppi, and Y. Xie, "Understanding the trade-offs in multi-level cell ReRAM memory design," in *DAC*, 2013.

[8] Y. Naruke, S. Mori, N. Hayasaka, T. Process, and T. Liu, "A 130.7 mm2 2-layer 32Gb ReRAM memory device in 24nm technology," in *ISSCC*, 2013.

[9] R. Fackenthal, M. Kitagawa, W. Otsuka, K. Prall, D. Mills, K. Tsutsui *et al.*, "A 16Gb ReRAM with 200MB/s write and 1GB/s read in 27nm technology," in *ISSCC*. IEEE, 2014.

[10] Intel, "Intel 3D XPoint Unveiled-The Next Breakthrough in Memory Technology," 2015. [Online]. Available: http://www.intel.com/content/www/us/en/architecture-and-technology/3d-xpoint-unveiled-video.html

[11] H. S. P. Wong, H. Y. Lee, S. Yu, Y. S. Chen, Y. Wu, P. S. Chen *et al.*, "Metal-oxide RRAM," *IEEE MICRO*, vol. 100, pp. 1951–1970, 2012.

[12] "International roadmap for semiconductors Report," 2012. [Online]. Available: http://www.itrs.net/ITRS_1999-2014_Mtgs,-Presentations-&-Links/2012ITRS/Home2012.htm

[13] J. L. Henning, "Performance counters and development of SPEC CPU2006," *ACM SIGARCH Computer Architecture News*, vol. 35, no. 1, p. 118, 2007.

[14] B. Jacob, S. Ng, and D. Wang, *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann, 2010.

[15] D. P. Bovet and M. Cesati, *Understanding the Linux kernel*. " O'Reilly Media, Inc.", 2005, no. November.

[16] L. E. Ramos, E. Gorbatov, and R. Bianchini, "Page placement in hybrid memory systems," in *ICS*, 2011.

[17] H. Y. Lee, Y. S. Chen, P. S. Chen, P. Y. Gu, Y. Y. Hsu, S. M. Wang *et al.*, "Evidence and solution of over-RESET problem for HfOX based resistive memory with sub-ns switching speed and high endurance," in *IEDM*, 2010.

[18] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," in *ISCA*, jun 2009.

[19] 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 *MICRO*, New York, New York, USA, dec 2009.

[20] 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," in *ISCA*, jun 2010.

[21] N. Binkert, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish *et al.*, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, p. 1, aug 2011.

[22] M. Poremba and Y. Xie, "NVMain: An Architectural-Level Main Memory Simulator for Emerging Non-volatile Memories," in *ISVLSI*. IEEE, 2012.

[23] J. D. McCalpin, "STREAM Benchmark." [Online]. Available: https://www.cs.virginia.edu/stream/

[24] S. Eyerman and L. Eeckhout, "System-level performance metrics for multiprogram workloads," *IEEE Micro*, vol. 28, no. 3, pp. 42–53, 2008.