SMART: On Simultaneously MArching RaceTracks to Improve the Performance of RaceTrack-based Main Memory

Xiangjun Peng
The Chinese University of Hong Kong
Hong Kong

Ming-Chang Yang*
The Chinese University of Hong Kong
Hong Kong

Ho Ming Tsui
The Chinese University of Hong Kong
Hong Kong

Chi Ngai Leung
The Chinese University of Hong Kong
Hong Kong

Wang Kang
Beihang University
Beijing

ABSTRACT

RaceTrack Memory (RTM) is a promising media for modern Main Memory subsystems. However, the "shift-before-access" principle, as the nature of RTM, introduces considerable overheads to the access latency. To obtain more insights for the mitigation of shift overheads, this work characterizes and observes that the access patterns, exhibited by the state-of-the-art RTM-based Main Memory, mismatches with the granularity of shift commands (i.e., a group of RaceTracks called Domain Block Cluster (DBC)). Based on the characterization, we propose a novel mechanism called SMART, which simultaneously and proactively marches all DBCs within a subarray, so that subsequent accesses to other DBCs can be served without additional shift commands. Evaluation results show that, averaged across 15 real-world workloads, SMART significantly outperforms other state-of-the-art proposals of RTM-based Main Memory by at least 1.53X in terms of the total execution time, on two different generations of RTM technologies.

ACM Reference Format:
Xiangjun Peng, Ming-Chang Yang, Ho Ming Tsui, Chi Ngai Leung, and Wang Kang. 2022. SMART: On Simultaneously MArching RaceTracks to Improve the Performance of RaceTrack-based Main Memory. In *Proceedings of 59th ACM/IEEE Design Automation Conference (DAC '22).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3489517.3530538

1 INTRODUCTION

Modern Main Memory systems, which are normally composed of Dynamic Random Access Memory (DRAM), suffer from the slowdown of scaling [5]. One promising alternative to DRAM is RaceTrack Memory (RTM), which can provide storage-class density, deliver memory-level access latency and even maintain low energy consumption [2, 12]. Currently there are two generations of RTM (i.e. from Domain-Wall (DW) RTM to Skyrmion (Sky) RTM). In RTM, information is encoded as a magnetic bits, and a RaceTrack contains a train of magnetic bits. Both generations of RTM share two types of operations. (1) access operation (i.e. read/write), to retrieve/change the data of the magnetic bit via the access port; and (2) shift operation, to shift a train of magnetic bits within a RaceTrack to map the demanded bit with the access port, to satisfy access operations.

Given the above characteristics, both generations of RTM share the "shift-before-access" principle, which requires shift operations to match the demanded magnetic bit with the access port before any access operations. Recent efforts are devoted to reduce the number of shifts for better performance, which can be categorized into two directions. One direction is to exploit key insights from the applications/algorithm to specialize for RTM (e.g., data placement strategies [3]) on RTM. The other direction is to exploit architectural characteristics to hide RTM access latency (e.g., profile-driven pre-shifting [1]). However, these solutions require extra information from the applications/algorithms to determine which RaceTracks to pre-shift, and therefore lacks the generality and applicability to fit in with the RTM-based architectural designs.

Speaking of RTM-based architectural designs, there is also a considerable amount of efforts to architect RTM as the alternatives to different components in the memory hierarchy. For instance, RTM has been considered as the alternatives to registers (e.g., [10]), caches (e.g., [8]), and storage (e.g., [11]). But only a limited amount of works focus on architecting RTM as Main Memory subsystems (e.g., [4]). To minimize shift overheads in RTM-based Main Memory, the state-of-the-art proposal exploits command reordering within the Memory Controllers to minimize the number of shifts within the current command queue [7]. However, this approach can only provide marginal benefits in terms of the performance and energy efficiency, due to the limited size of the command queue.

To obtain more insights to develop effective mechanisms for the mitigation of shift overheads, we characterize the state-of-the-art RTM-based Main Memory [7] on 15 real-world workloads. In the state-of-the-art RTM-based Main Memory, the granularity of access/shift commands in RTM-based Main Memory is at a unit of a Domain Block Cluster (DBC), which refers to a group of RaceTracks. Based on our characterization, we make the key observation that RTM-based Main Memory exhibits a high extent of access patterns on all magnetic bits at the same offset of all DBCs within a subarray, and these bits can be logically abstracted as a row. We quantitatively identify the mismatch between DBC-based commands and row-level access patterns. More specifically, we observe that *a recently-accessed row is likely to be accessed consecutively.* Averaged across 15 workloads of our characterization, 82.28% of the shift commands

*Corresponding Author.

DAC '22, July 10–14, 2022, San Francisco, CA, USA

ACM ISBN 978-1-4503-9142-9/22/07…$15.00
https://doi.org/10.1145/3489517.3530538

to a subarray are performed for consecutive accesses to different DBCs associated with the same row.

Based on the above characterization, we propose SMART, a mechanism to deliver higher performance than the state-of-the-art RTM-based Main Memory [7]. The key idea of SMART is to *simultaneously and proactively march all RaceTracks associated with the same row, instead of shifting only RaceTracks of the requested DBC*. In this way, further accesses to other DBCs, associated with the same row, can be served without additional shift commands, which reduce the overall execution time and substantially achieve better performance. Evaluation results show that SMART can deliver significant performance improvements than other state-of-the-art proposals of RTM-based Main Memory [7]. Averaged across 15 real-world workloads, SMART significantly outperforms other state-of-the-art proposals of RTM-based Main Memory [7] by at least 1.53X in terms of the total execution time, under two different generations of RTM technologies. In addition, since SMART shifts all DBCs within a subarray at a time, a shift command in SMART consumes more energy than a shift command in the state-of-the-art RTM-based Main Memory [7]. However, evaluation results confirm that SMART still reduces the overall energy consumption by at least 33.74% on average.

The rest of this paper is organized as follow. § 2 provides background of RTM and RTM-based Main Memory. § 3 presents our characterization of the state-of-the-art RTM-based Main Memory on 15 real-world workloads, as the motivation. § 4 elaborates detailed designs and implementations of our proposed mechanism SMART. § 5 describes our experimental methodology, and § 6 presents the experimental results.

2 BACKGROUND

2.1 Basics of RaceTrack Memory

RaceTrack Memory (RTM) is first proposed in 2008 by IBM [12], which can provide storage-class density, deliver memory-level access latency and even maintain low energy consumption [2, 12]. Currently there are two generations of RTM. The first generation of RTM encodes information in a magnetic nanoribbon by a train of magnetic domains, which can be spun up as the bit "1" and spun down as the bit "0". These domains are separated by domain walls (DWs). We denote this generation of RTM as DW-RTM. The second generation of RTM leverages magnetic particle-like spin textures, called Skyrmions, to carry information, which explicitly interpret the presence of a Skyrmion as the bit "1" and the absence of a Skyrmion as the bit "0". We denote this generation of RTM as Sky-RTM.

In both generations of RTM, information is encoded as a magnetic bits, and a RaceTrack contains a train of magnetic bits. Therefore, both generations of RTM share the "shift-before-access" principle, which requires shift operations to match the demanded magnetic bit with the access port before any access operations. In both generations of RTM, there are two common types of operations. (1) access operation (i.e. read/write), to retrieve/change the data of the magnetic bit via the access port; and (2) shift operation, to shift a train of magnetic bits within a RaceTrack to map the demanded bit with the access port, to satisfy access operations.

2.2 Architectural Organization for RTM-based Main Memory

To efficiently organize RTM into arrays, prior works introduce the concept of *macro units* to improve the area efficiency in the designs of RTM arrays. A *macro unit* refers to a group of RaceTracks, and there are multiple *macro units* within an array [14]. This design choice aims to achieve high capacity of RTM-based Main Memory, because a transistor (i.e., to organize drivers for RTM) is normally wider than a RaceTrack. Therefore, a *marco unit* contains multiple RaceTracks, and one or more *marco units* share a single driver. Therefore, all RaceTracks within one or more *macro unit*, controlled by a single driver, operate in a lock-step manner (i.e. one issued operation triggering all RaceTracks to perform the same operation).

The state-of-the-art proposal of RTM-based Main Memory (i.e., [4, 7]) is highly similar to commodity DRAM architectures (i.e., [5]) in a hierarchical manner. Each chip of RTM-based Main Memory is organized as follow. Each RTM chip can be organized by one or more RTIM rank(s). A RTM rank consists of multiple RTM banks, as shown in Figure 1-(A), which share a global Chip I/O; each RTM bank consists of multiple RTM subarrays, as shown in Figure 1-(B). The separation of multiple RTM subarrays increases the total capacity of each RTM bank. The outstanding difference between RTM-based Main Memory and DRAM lies on the internal organization of a subarray. Each RTM subarray contains multiple Domain Block Clusters (DBCs) and a driver, which provide separate controls for different DBCs (shown in Figure 1-(C)). Finally, within each DBC, multiple RaceTracks are aligned vertically and they work in a lock-step manner (i.e. one issued command triggering all RaceTracks to perform the same command).

In the state-of-the-art RTM-based Main Memory [7], each RTM rank typically consists of 8 RTM banks. In each RTM bank, there are 128 RTM subarrays. Each RTM subarray typically consists of 512 rows, and consists of 16 DBCs. Each DBC typically consists of a number of RaceTracks, which equals to the size of a word/cacheline. RTM commands operate at a granularity of a single DBC, therefore all RaceTracks in a particular DBC operate in a lock-step fashion so that a word/cacheline is interleaved across different RaceTracks within a DBC to satisfy simultaneous access for all bits.

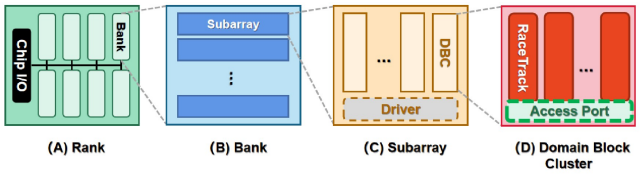


Figure 1: Architectural organization of RTM-based Main Memory: (A) Rank, (B) Bank, (C) Subarray and (D) Domain Block Cluster.

2.3 Command Handling in RTM-based Main Memory

To perform accesses to RTM-based Main Memory, the state-of-the-art proposal exploits Memory Controllers to bridge CPU requests and RTM commands [7]. The Memory Controller in CPUs buffers CPU requests (i.e., read/write) in a transaction queue. Subsequently,

each transaction is converted into a set of RTM commands which are placed in a command queue. In RTM-based Main Memory, there are three types of commands: (1) read commands are used to read data from a certain DBC; (2) write commands are used to write data to a certain DBC; and (3) shift commands are used to map access ports with a certain DBC.

The Memory Controller further dispatches RTM commands to DBCs, one at a time, as follows. ❶ The Memory Controller needs to perform address decoding to map commands with the target positions. In the-state-of-the-art RTM-based Main Memory [7], address decoding performs at the granularity at DBC-level. More specifically, each address decoding needs to translate the address tuple (rank, bank, subarray, DBC), to serve the incoming commands. ❷ Due to the "shift-then-access" principle, the Memory Controllers for RTM-based Main Memory need to insert shift commands, based on the command queue. Hence, the Memory Controller maintains the positions of all access ports corresponding to each RaceTrack, using a position tracking table for updating the changes. In the default data layout, RaceTracks are grouped into DBCs and all ports in a DBC move together in the lock-step fashion. This implies that the port positions of all RaceTracks in a DBC are always the same. ❸ The Memory Controller schedules and issues RTM commands to different memory banks, while respecting both timing and flow of commands constraints. Once a command is issued, respective sanity checks (for timing constraints) are performed. At completion of requests, requests are removed from the Memory Controller.

2.4 Shift-Minimized Command Reordering in RTM-based Main Memory

The permutation of access commands can result in a different number of shift commands. To minimize shift overheads, the state-of-the-art RTM-based Main Memory applies shift-minimized reordering within the command queue [7]. In this manner, all commands within the command queue are reordered, to achieve the minimum amount of required shift commands. However, as revealed by § 6, this approach for reducing shift commands has a considerable limitation: the reordering mechanism can only provide marginal benefits in terms of both the performance and energy efficiency, because it's limited by the size of the command queue.

3 MOTIVATION

To obtain more insights for the mitigation of shift overheads, we leverage our experimental infrastructure (i.e., described in § 5) to examine the access patterns exhibited by RTM-based Main Memory on 15 real-world workloads. Note that all magnetic bits at the same offset of all DBCs within a subarray, and these bits can be logically abstracted as a row. Figure 2 quantitatively reports the ratio of consecutive shift commands, which are performed for accessing different DBCs associated with the same row, to all shift commands for each of the subarrays. We make the key observation that, on average, the access patterns majorly exhibit at row level instead of DBC level. Therefore, the state-of-the-art RTM-based Main Memory design [4, 7] results in a huge amount of serialized shift commands, which are required to serve requests for different DBCs associated with the same row. Specifically, 82.28% of the shift commands to a subarray are performed for consecutive accesses to different DBCs associated with the same row.

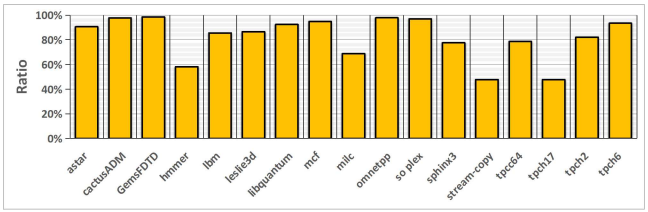


Figure 2: Ratio of consecutive shift commands, which are performed for accessing different DBCs associated with the same row, to all shift commands for each of the subarrays.

The above phenomenon can be attributed to the architectural design of the state-of-the-art RTM-based Main Memory [7], which makes a single shift command only drives the RaceTracks from a single DBC at a time. Therefore, there exists a mismatch between (i) the granularity of RTM access/shift commands (i.e., one DBC at a time) and (ii) the access patterns at the level of a row within a subarray. Therefore, to make RTM-based Main Memory more efficient, it's essential to design effective mechanisms to resolve the mismatch between the granularity of RTM commands and the exhibited row-level access patterns.

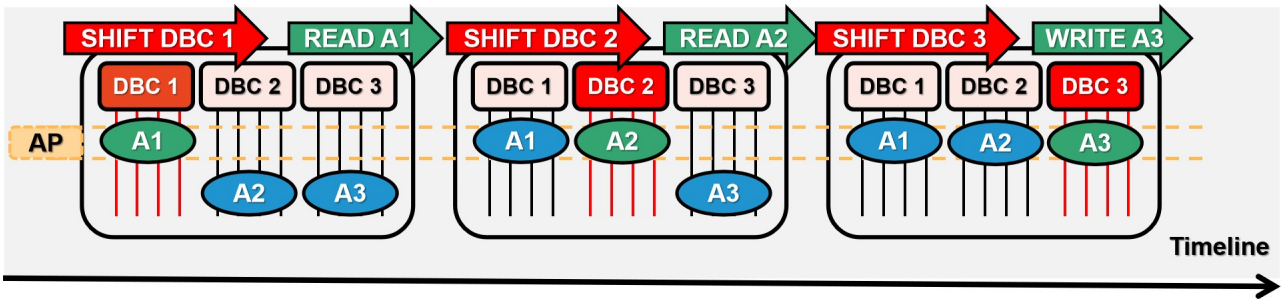
4 SMART: DESIGN & IMPLEMENTATION

Based on the above characterization, this section introduce SMART, a novel mechanism to improve the performance of RTM-based Main Memory. This section first covers the design of SMART via exploring the overall design and providing a concrete example. Then this section elaborates the details regarding the implementation of SMART.

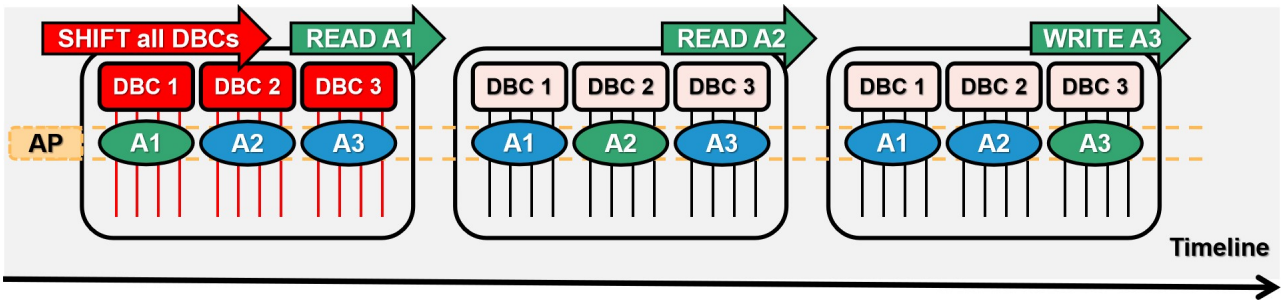
4.1 The Design of SMART

As demonstrated in Section 3, existing RTM-based Main Memory lacks efficient mechanisms to exploit the row-level access patterns. In current designs, these patterns cannot be effectively exploited because every shift command is targeted to a particular DBC (i.e., word-/cahceline-size blocks). In this manner, each shift command drives all RaceTracks in a specific DBC at a time. Thus, if a series of consecutive accesses to different DBCs associated with the same row, it's needed to drive each DBC separately, to shift RaceTracks to match with the access ports.

The key idea of SMART is to issue a shift command on all DBCs within a subarray at a time, so that subsequent accesses to other DBCs can be served without additional shift commands, which reduce the overall execution time and substantially achieve better performance. More specifically, SMART aims to proactively parallelize as many shift commands as possible by exploiting our characterized access patterns: instead of targeting one specific DBC, SMART issues one shift command to drive all DBCs within a subarray, and these bits can be logically abstracted as a row. Therefore, a shift command in SMART shifts a row at a time, rather than a DBC at a time.



(A) The state-of-the-art approach to handle the example memory requests (six commands in total)



(B) Our proposal "SMART" to handle the example memory requests (four commands in total)

Figure 3: A comparative example to understand the benefits of SMART.

To demonstrate the benefits of SMART, we use an illustrative example sequence of memory requests. and there are three consecutive accesses to a row within a subarray: ❶ READ access to the DBC 1 of Row A; ❷ READ access to the DBC 2 of Row A; and ❸ WRITE access to the DBC 3 of Row A. Hereby, we assume all DBCs at initial state and they are not mapped with the access port.

As shown in Figure 3A, the state-of-the-art RTM-based Main Memory system [7] handles the example sequence of memory requests with six commands. ❶ Shift the DBC 1 of Row A to the access port; ❷ READ access to the DBC 1 of Row A; ❸ Shift the DBC 2 of Row A to the access port; ❹ READ access to the DBC 2 of Row A; ❺ Shift the DBC 3 of Row A to the access port; and ❻ WRITE access to the DBC 3 of Row A. By contrast, Figure 3B shows how SMART handles the example sequence of memory requests with only four commands. ❶ Shift all DBCs of Row A to the access port (i.e., DBC 1,2,3); ❷ READ access to the DBC 1 of Row A; ❸ READ access to the DBC 2 of Row A; and ❹ WRITE access to the DBC 3 of Row A. In summary, "READ A2" and "WRITE A3" can be directly served without addition shift commands in SMART, which brings performance benefits.

4.2 Implementation Details

To implement SMART, existing structures of RTM-based Main Memory can be reused [7]. For SMART, there are three major parts to be considered, including: (1) Address Decoding; (2) Shift Broadcasting; and (3) Position Tracking Table.

Address Decoding. To serve shift commands, the state-of-the-art RTM-based Main Memory [7] needs to decode the full address tuple. This is because a shift command needs to specify which rank, bank, subarray, row and DBC, to ensure the shift commands are performed correctly. In the design of SMART, a shift command only needs to specify which rank, bank, subarray and row, since the smallest granularity of shift command is a row. Therefore, address decoding in SMART can be more efficient in terms of both the implementation complexity and the performance.

Shift Broadcasting. In the state-of-the-art RTM-based Main Memory [7], a shift command is issued to drive only a single specific DBC. This is because all commands in RTM-based Main Memory operates at the granularity of a single DBC. As for the design of SMART, a shift command is issued to shift all RaceTracks from all DBCs within a subarray. To perform such a command simultaneously, we can exploit existing resources to broadcast the shift command, denoted as Shift Broadcasting. To support Shift Broadcasting, the implementations is relatively straightforward: SMART only needs an unified control of all separate DBCs, instead of individual controls for each DBC within a subarray.

Position Tracking Table. In the state-of-the-art RTM-based Main Memory [7], the position of every DBC needs to be recorded and updated according to the shift commands. In the state-of-the-art RTM-based Main Memory [7], the granularity of a shift command is at the level of a DBC. Therefore, the Memory Controller needs to keep RaceTrack for the current positions of all DBCs, as well as which hierarchical components they are associated with (i.e. rank, bank, subarray). This is because each DBC can have asynchronous positions as the program executes, in terms of the state-of-the-art RTM-based Main Memory [7]. In the design of SMART, the granularity of a shift command is at the level of a row (i.e. all DBCs within a subarray). Therefore, the Memory Controller only needs to keep RaceTrack for the current positions of a row, since all DBCs within a subarray have synchronous positions with SMART.

5 EXPERIMENTAL METHODOLOGY

5.1 Simulated System Configurations

Table 1 lists the configuration of the default system. To evaluate the performance and energy consumption of SMART, we use a cycle-accurate RaceTrack-based Main Memory simulator, RTSim [7] in CPU trace-driven mode. RTSim is extended from a mainstream Non-Volatile Memory simulator, NVMain [13]. We collect the CPU traces are using Pin [9]. For energy consumption, we use the integrated module of RTSim[7] for energy measurement. We model the Memory Controller using (1) the default setting within RTSim, which relies on First-Come-First-Serve (FCFS) policy; and (2) shift-minimized command reordering within RTSim, which schedules the command queue with the minimal number of shift commands.

Table 1: Default simulated system configurations.

Processor	1 core, 4GHz clock frequency; 3-wide issue, 8 MSHRs/core; 128-entry instruction window
Last-Level Caches	64B cache-line, 16-way associative, 4MB cache size
Memory Controller	64-entry read/write request queues; FCFS scheduling [4, 7];
RTM-based Main Memory	1 channel; 1 rank/channel; 8 banks/rank; 64K rows/bank
❶ Domain-Wall RTM Config.	8KB Row Size; 64B DBC Size, 128 DBCs; Latency & Energy based on [7]
❷ Skyrmion RTM Config.	16KB Row Size; 128B DBC Size, 128 DBCs; Latency & Energy based on [6]

In total, eight different configurations of RTM-based Main Memory are considered for the evaluations. Firstly, two different generations of RTM technologies, including Domain Wall RTM (denoted as DW-RTM) and Skyrmion RTM (denoted as Sky-RTM) are considered. For different generations of RTM technologies, we varies the configurations for the internal structure of a subarray. Restricted by the state-of-the-art RTM-based Main Memory [7], we comparatively model DW-RTM and Sky-RTM based on the most recent study [6]. This study suggests Sky-RTM can contain more RaceTracks than DW-RTM within a single DBC (i.e., approximately 2X to 4X more than DW-RTM). Secondly, four different configurations for the reduction of shift commands, including the baseline FCFS scheduling, shift-minimized command reordering, SMART and the combinations of reordering and SMART. The configurations for different shift-oriented optimizations are listed as follows.

- **Baseline** follows the default configuration of RTM-based Main Memory using DW-RTM or Sky-RTM,
- **Reordering** denotes shift-minimized command reordering on the default configuration of RTM-based Main Memory using DW-RTM or Sky-RTM.
- **SMART** denotes our proposed mechanism on RTM-based Main Memory using DW-RTM or Sky-RTM.
- **SMART + Reordering** denotes the combination of shift-minized command reordering and our proposedSMART on RTM-based Main Memory using DW-RTM or Sky-RTM.

For evaluation metrics, we measure the execution time and report the speedup by normalizing different settings to the Baseline, and measure and report the energy consumption by normalizing energy consumption to the Baseline.

5.2 Selected Workloads

We select 15 real-world workloads from representative benchmarks, including SPEC2006 and TPC-H. More specifically, we use (1) SPEC2006 Integer Benchmark, including *astar*, *hmmer*, *libquantum*, *mcf*, *omnetpp*; (2) SPEC2006 Floating-Point Benchmark, including *cactusADM*, *GemsFDTD*, *lbm*, *leslie3d*, *milc*, *so plex*, *sphinx3*. and (3) TPC-H benchmark, including *tpch17*, *tpch2*, *tpch6*.

6 EXPERIMENTAL RESULTS

6.1 Performance Improvements

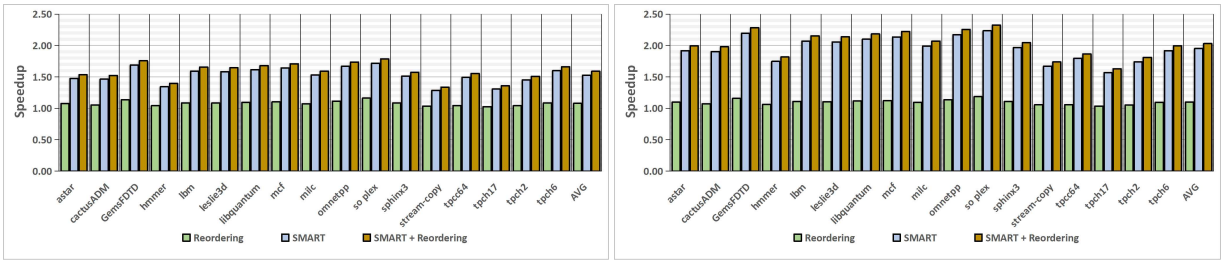
Figure 4 report the performance improvements provided by Reordering, SMART and the combination of these two, over the Baseline. Figure 4A reports the results when architecting RTM-based Main Memory using DW-RTM, and Figure 4B reports the results when architecting RTM-based Main Memory using Sky-RTM.

We make three observations. First, SMART significantly outperforms other state-of-the-art RTM-based Main Memory designs. Averaged across 15 real-world workloads, SMART achieves at least 1.52X and 1.42X speedup (using DW-RTM), compared with the Baseline and Reordering. This is because SMART can significantly reduce the overall amount of shift commands. Second, the significant benefits of SMART are consistent in different generations of RTM technologies. Averaged across 15 real-world workloads, SMART achieves at least 1.95X and 1.78X speedup (using Sky-RTM), compared with the Baseline and Reordering. This is because (1) the design of SMART exploits the characteristics at the architectural level; and (2) Sky-RTM can increase the number of DBCs within a subarray, which enhances the access patterns. Third, the benefits of SMART are consistent in different configurations of RTM-based Main Memory. Averaged across 15 real-world workloads, SMART + Reordering achieves at least 1.58X, 1.47X and 1.04X speedup (using Sky-RTM), compared with the Baseline, Reordering and SMART. This is because the designs and implementations of SMART only requires the negligible changes to the existing design.

6.2 Energy Savings

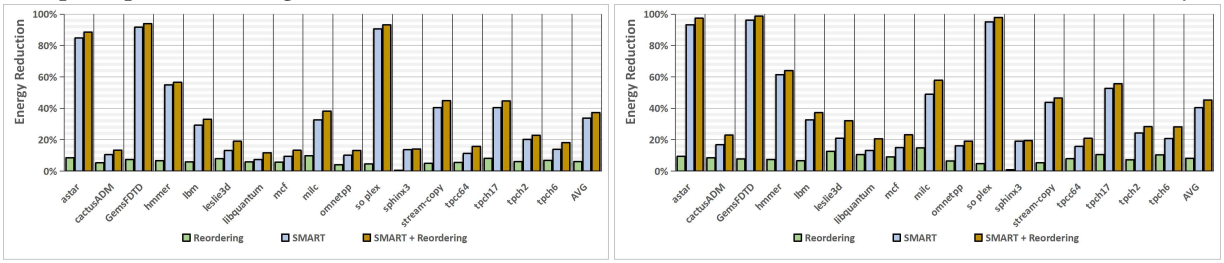
Since SMART shifts all DBCs within a subarray at a time, a shift command consumes more energy than a shift command in the state-of-the-art RTM-based Main Memory [7]. Hence, it's important to examine the overall energy consumption of SMART. Figure 5 report the energy savings provided by Reordering, SMART and the combination of these two, over the Baseline. Figure 5A reports the results when architecting RTM-based Main Memory using DW-RTM, and Figure 5B reports the results when architecting RTM-based Main Memory using Sky-RTM.

We make two observations. First, SMART *always* saves more energy than other state-of-the-art RTM-based Main Memory designs, regardless of different RTM generations. Averaged across 15 real-world workloads, SMART reduces the energy consumption by at least 33.74% and 17.98% (using DW-RTM), compared with the Baseline and Reordering. This is because SMART can significantly reduce the execution time. Second, SMART provides more energy benefits in Sky-RTM, compared with DW-RTM. Averaged across 15 real-world workloads, SMART reduces the energy consumption by at least 40.32% and 20.31% (using Sky-RTM), compared with the Baseline and Reordering. This is because the SMART can bring more performance benefits in Sky-RTM, compared with DW-RTM.



(A) Domain Wall RTM (B) Skyrmion RTM

Figure 4: Speedup of Reordering, SMART and their combination over the state-of-the-art RTM-based Main Memory design.



(A) Domain Wall RTM (B) Skyrmion RTM

Figure 5: Energy reduction of Reordering, SMART and their combination over the state-of-the-art RTM-based Main Memory design.

7 CONCLUSION

We present SMART, a mechanism to deliver higher performance RTM-based Main Memory systems. The key idea of SMART is to exploit the characterized row-level access patterns, to simultaneously march all RaceTracks associated with the same row. SMART can effectively exploit the observed access patterns of RTM-based Main Memory system, by pre-shifting those within-the-same-row RaceTracks (i.e., which are very likely to be accessed later). We provide concrete details for the implementations of SMART, and we show that SMART is a cost-effective approach. Our evaluations show that, compared with the existing design, SMART can significantly improve both the performance and energy efficiency across 15 real-world workloads, and the benefits are consistent in both generations of RTM technologies.

ACKNOWLEDGEMENT

We thank all reviewers from DAC'22 for their constructive and valuable feedback. This work is supported in part by The Research Grants Council of Hong Kong SAR (Project Nos. CUHK14210320 and CUHK14208521), the National Natural Science Foundation of China (No. 61871008), and Beijing Nova Program from Beijing Municipal Science and Technology Commission (No. Z201100006820042 and No. Z211100002121014).

REFERENCES

[1] Ehsan Atoofian. 2015. Reducing shift penalty in Domain Wall Memory through register locality. In *2015 International Conference on Compilers, Architecture and Synthesis for Embedded Systems, CASES 2015, Amsterdam, The Netherlands, October 4-9, 2015*, Ravi Iyer and Siddharth Garg (Eds.). IEEE, 177–186. https://doi.org/10.1109/CASES.2015.7324558

[2] Albert Fert, Vincent Cros, and Joao Sampaio. 2013. Skyrmions on the track. *Nature nanotechnology* 8, 3 (2013), 152–156.

[3] Yun-Shan Hsieh, Po-Chun Huang, Ping-Xiang Chen, Yuan-Hao Chang, Wang Kang, Ming-Chang Yang, and Wei-Kuan Shih. 2020. Shift-Limited Sort: Optimizing Sorting Performance on Skyrmion Memory-Based Systems. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 39, 11 (2020), 4115–4128. https://doi.org/10.1109/TCAD.2020.3012880

[4] Qingda Hu, Guangyu Sun, Jiwu Shu, and Chao Zhang. 2016. Exploring Main Memory Design Based on Racetrack Memory Technology. In *Proceedings of the 26th edition on Great Lakes Symposium on VLSI, GLVLSI 2016, Boston, MA, USA, May 18-20, 2016*. ACM, 397–402. https://doi.org/10.1145/2902961.2902967

[5] Bruce Jacob, David Wang, and Spencer Ng. 2010. *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann.

[6] Wang Kang, Bi Wu, Xing Chen, Daoqian Zhu, Zhaohao Wang, Xichao Zhang, Yan Zhou, Youguang Zhang, and Weisheng Zhao. 2020. A Comparative Cross-layer Study on Racetrack Memories: Domain Wall vs Skyrmion. *ACM J. Emerg. Technol. Comput. Syst.* 16, 1 (2020), 2:1–2:17. https://doi.org/10.1145/3333336

[7] Asif Ali Khan, Fazal Hameed, Robin Bläsing, Stuart Parkin, and Jeronimo Castrillon. 2019. RTSim: A cycle-accurate simulator for racetrack memories. *IEEE Computer Architecture Letters* 18, 1 (2019), 43–46.

[8] Bing Li, Fan Chen, Wang Kang, Weisheng Zhao, Yiran Chen, and Hai Li. 2018. Design and Data Management for Magnetic Racetrack Memory. In *IEEE International Symposium on Circuits and Systems, ISCAS 2018, 27-30 May 2018, Florence, Italy*. IEEE, 1–4. https://doi.org/10.1109/ISCAS.2018.8351681

[9] Chi-Keung Luk, Robert S. Cohn, Robert Muth, Harish Patil, Artur Klauser, P. Geoffrey Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim M. Hazelwood. 2005. Pin: building customized program analysis tools with dynamic instrumentation. In *Proceedings of the ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation, Chicago, IL, USA, June 12-15, 2005*. ACM, 190–200. https://doi.org/10.1145/1065010.1065034

[10] Mengjie Mao, Wujie Wen, Yaojun Zhang, Yiran Chen, and Hai Li. 2017. An Energy-Efficient GPGPU Register File Architecture Using Racetrack Memory. *IEEE Trans. Computers* 66, 9 (2017), 1478–1490. https://doi.org/10.1109/TC.2017.2690855

[11] Eunhyuk Park, Sungjoo Yoo, Sunggu Lee, and Hai Helen Li. 2014. Accelerating graph computation with racetrack memory and pointer-assisted graph representation. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2014, Dresden, Germany, March 24-28, 2014*, Gerhard P. Fettweis and Wolfgang Nebel (Eds.). European Design and Automation Association, 1–4. https://doi.org/10.7873/DATE.2014.172

[12] Stuart SP Parkin, Masamitsu Hayashi, and Luc Thomas. 2008. Magnetic domain-wall racetrack memory. *Science* 320, 5873 (2008), 190–194.

[13] Matthew Poremba, Tao Zhang, and Yuan Xie. 2015. NVMain 2.0: A User-Friendly Memory Simulator to Model (Non-)Volatile Memory Systems. *IEEE Comput. Archit. Lett.* 14, 2 (2015), 140–143. https://doi.org/10.1109/LCA.2015.2402435

[14] Hongbin Zhang, Chao Zhang, Qingda Hu, Chengmo Yang, and Jiwu Shu. 2018. Performance analysis on structure of racetrack memory. In *23rd Asia and South Pacific Design Automation Conference, ASP-DAC 2018*, Youngsoo Shin (Ed.). IEEE, 367–374. https://doi.org/10.1109/ASPDAC.2018.8297351