Rethinking Data Structure Designs for Racetrack Memories

Hsin-Ting Chou*§, Chia-Wei Hsu*§, Ting-Syuan Lin*§, Chun-Chih Lai*, and Po-Chun Huang*‡
*National Taipei University of Technology

***Abstract*—Recently, *racetrack memories (RMs)* have significantly changed the existing computation models based on DRAM main memory. However, although RMs provide ideal storage density, access performance, and nonvolatile data storage, the intrinsic peculiarities of RMs create barriers that prevent directly using RMs as the medium of the main memory. Specifically, data bits must be shifted along the RM's nanotracks to align with arbitrary access ports, before they can be read out from the RMs. Thus, the data bits that are far away from any access ports are slow to access, resulting in biased worst-case access performance of different data bits on the nanotrack. To overcome this problem, existing data structures and algorithms must be redesigned to minimize the random-hopped memory accesses, so as to alleviate the performance overheads due to the massive shift operations of RMs. In this work, we revisit the designs of popular data structures, and discuss several potential techniques—either existing or novel—to make existing data structure and algorithm designs friendly on RMs. We believe that such an early investigation would be helpful for broadening the future application scenarios of RMs.**

1. Introduction

Classical memory and storage media, such as DRAM, NAND flash memory, and mechanical hard disk, have confronted serious challenges in overcoming their intrinsic limitations, such as the volatile storage capability, unsatisfying access performance, or limited device endurance. Thus, modern *nonvolatile memories (NVMs)* have emerged as energy-efficient alternatives to classical memory and storage media [1]. Such NVMs include the *phase-change memory (PCM)* [2], *spin-torque-transfer random-access memory (STT-RAM)* [3], *resistive memory (ReRAM)* [4], and *racetrack memories (RMs)* [5]. However, the special inherent characteristics and operational constraints of NVMs often create barriers which hinder the wider adoption of NVMs in real-world application scenarios.

Among the popular NVMs, RMs are considered a competitive choice as persistent storage devices, due to the satisfactory access performance, nonvolatile data storage capability, and excellent storage density, which is defined as the amount of data that can be stored in the same physical area or volume. There are two flavors of RMs: *domain-wall racetrack memories (DW-RMs)* [5], [6] and more-recent *skyrmion racetrack memories (SK-RMs)* [5], [7]. Because SK-RMs deliver better storage density and the unique abilities to randomly insert data bits to, or delete data bits from, the midway of existing bit streams, they are considered a more promising solution than DW-RMs in the future.

Despite the many advantages of SK-RMs, however, a major issue of SK-RMs is that the to-be-accessed data bits must be shifted along the *nanotrack* of SK-RM to align with any access port, before they can be retrieved from, or updated in situ, in the SK-RM [8]. As a result, the data bits that are far away from any access ports take longer time to be accessed, resulting in very unstable worst-case access latencies. In addition, the introduction of SK-RMs also brings in extra data errors, which occur when many consecutive bit-0s are stored on the nanotrack or when a long-distance shift is performed [9]. Architecture-level solutions are therefore in urgent need to resolve the access performance and data reliability problems of SK-RMs. On the other hand, these hardware peculiarities of SK-RMs have considerably changed the computation model. Consequently, existing data structures and algorithms originally designed for the *random-access machine model (RAM model)* [10] need to be modified to fit a new *racetrack-memory model (RM model)*, so as to optimally reveal the benefits of SK-RMs. Motivated by the many observations above, this work first revisits the RM model first presented in [8], and discusses several potential techniques—existing or novel—for optimizing existing data structures and algorithms on SK-RMs. We believe that these techniques can shed light on establishing a brand-new computing ecosystem based on SK-RMs, which is important for the performance-demanding but energy-critical computing environments in the future.

The rest of this paper is organized as follows. First of all, Section 2 presents the shifts of computation models, from the existing RAM model to the novel RM model. After that, Section 3 discusses several potential techniques for revising existing data structure and algorithm designs for SK-RMs. Section 4 then concludes this work.

2. The Shifts of Computation Models: From DRAM to Racetrack Memories

2.1. Overview

Computation models are a crucial factor that should be carefully concerned when designing data structures and algorithms. While there are diversified existing computation models from the perspectives of different hardware components, we concentrate on *main-memory models*, which is especially important for modern I/O intensive applications. (Models of other hardware components such as parallel processors or network connections also exist, but are beyond the scope of this paper.) Specifically, we consider the *random-access machine model (RAM model)* [10], which is closer to the characteristics of DRAM main memories, and then discuss a novel *racetrack-memory based model (RM model)*, which considers the *shift operations* and *multi-nanotrack, multi-access-port parallel architecture* of SK-RMs.

§. The first three authors made equal contributions to this work.
‡. Po-Chun Huang is the corresponding author. Email: po.chun.huang@mail.ntut.edu.tw.

2.2. Random-access Machine (RAM) Model

The *random-access machine model (RAM model)* is the classical computation model that are widely adopted in the designs of existing data structures, such as search trees and hash tables [10]. *If caches and buffers can be ignored*, the RAM model well approximates a main memory based on DRAM, which respects the following characteristics:

1) Accesses (reads or writes) are done in the minimal unit of a *machine word*, which is typically 32 or 64 bits, according to the CPU architecture. Thus, accessing even a bit of data requires to access the whole machine word, causing the *read/write amplification* problem [11].
2) Each variable of primitive types, such as a lexical character, integer, floating-point number, and pointer, takes one machine word of memory space.
3) Supported memory access operations include: *read* (retrieve a machine word without modifying it) and *write* (overwrite a machine word).
4) Accessing (reading or writing) a machine word takes one unit of time, despite of the address of the machine word in memory.
5) The caching and buffering are ignored. The multi-bank memory chip organization is often ignored as well, implying the classical *uniform memory access (UMA)* architecture [12]. (For large-scale servers or even server clusters, the *nonuniform memory access (NUMA)* architecture [13] should be considered. However, this is orthogonal issue to our discussions in this paper, because the main aim of this paper is to consider the model shifts due to the changes of the memory media.)

2.3. Racetrack-memory (RM) Model [8]

The computation model based on RMs, namely the *racetrack-memory based model (RM model)* is quite different from the RAM model. Since *skyrmion racetrack memories (SK-RMs)* have emerged to replace the classical *domain-wall racetrack memories (DW-RMs)* [5], [6], our RM model is essentially based on SK-RMs. With the RAM model, each data bit is directly stored in a memory cell at a fixed position. In contrast, with the RM model, the data bits are stored on a nanotrack and need to be moved by the time-costly *shift operation* to align with any access port at a fixed position, so as to be accessed. As a result, if there is only one access port on each nanotrack, the RM works just like a *tape*, where sequential accesses are preferred over random ones, due to the lower shift overheads. The RM model is as follows [8]:

1) Each RM chip has a number n_t of equal-size *nanotracks*, each of which has n_p equally-separated fixed *access ports* on it. The distance between adjacent access ports is d bits, indicating that at most d *skyrmions* (*i.e.*, bit-1s) or *nonskyrmions* (*i.e.*, bit-0s) can be stored in the space between adjacent access ports. We (somewhat arbitrarily) assume that a skyrmion or a nonskyrmion takes the same amount of space on the nanotrack. Thus, each nanotrack can store up to $d \times n_p$ bits of data. We further assume sufficient overhead areas are preserved so that every data bit can be accessed without shifting any bits past the beginning or the end of the nanotrack.
2) The data bits on a nanotrack must be shifted so that the to-be-accessed data bit can be aligned with any access port to be read. Since there is an access port for every d bits, the best- and worst-case shift overheads to access a bit of data are 1 and $\lceil d/2 \rceil$ shifts, respectively. If we enforce unidirectional shifts during data accesses, the worst-case shift overhead to access a bit of data is $(d-1)$ shifts. (The data bits on the nanotrack must be shifted back after each access operation.)
3) The multiple access ports of different nanotracks can be accessed in parallel. Thus, each nanotrack can be regarded as an independent memory part that can be accessed without affecting each other. However, due to the bandwidth limitation of the shared data bus, it is often necessary to assume that the number of parallel-accessible nanotracks is bounded by a threshold γ. To make the existing *bit-interleaved* and *block cluster* [14] data placement policies work, the value of γ should be no fewer than the number of bits in a machine word, such as 32 or 64 on generic desktop computers nowadays. The designs of the data placement policies should therefore concern the parallel access capabilities of access ports on different nanotracks, so as to maximize the throughput of data accesses.
4) The multiple access ports on the same nanotrack cannot perform inserts or deletes in parallel, due to the *port synchronization* and *data alignment problems* of data bits and access ports. Since all data bits on the same nanotrack are shifted at once, performing an insert or delete operation on one access port of a certain nanotrack must lock any other parallel accesses, including insert, delete, read, and write on other access ports of the same nanotrack.
5) The multiple access ports on the same nanotrack can perform read or write operations in parallel, since reads and writes do not affect the positions of data bits on the nanotrack, thereby avoiding the port synchronization and data alignment problems. To the knowledge of the authors, there are currently no real-world implementations of SK-RMs with this functionality, however we anticipate that such a feature will be available in the future. Nevertheless, if the degree of *inter-nanotrack parallelism* is sufficient, the *intra-nanotrack parallelism* might be unimportant.
6) Supported memory access operations of SK-RMs [8]:
   a) *Read*: Retrieve a bit of data currently aligned with an access port without modifying it. To read a whole variable, we have to read all its bits from the SK-RM. Thus, the time to read a variable is affected by the *data placement policies*, which determine how the different data bits of a variable are placed on the nanotracks of the SK-RMs.
   b) *Write*: Overwrite a data bit currently aligned with an access port. Similarly, to write a whole variable, all its bits need to be updated in the worst case, and data placement policies matter.
   c) *ShiftLeft(k)*: Move all data bits, including *skyrmions* (each of which typically represents a bit-1) and *nonskyrmions* (each of which typically represents a bit-0) left by k bit positions on the nanotrack. Although somewhat imprecise, we assume that a skyrmion takes the same width of space on the nanotrack, as a nonskyrmion does. Alternatively, we can assume that hardware can correctly perform shift operations even when a skyrmion and a nonskyrmion take different widths of space on the nanotrack.

d) *ShiftRight(k)*: The same as *ShiftLeft*, except for that all bits are shifted right by k bit positions.
e) *InsertLeft(b)* and *InsertRight(b)*: Move all data bits to the left/right of the current access port left/right by one bit position, and insert a new bit b of data to the space created by the left shift operation. Note that creating a skyrmion takes more time and energy than creating a nonskyrmion.
f) *DeleteLeft(b)* and *DeleteRight(b)*: Remove and return the data bit aligned with the target access port, and move all data bits to the right/left of the current access port left/right by one bit position, which effectively removes the space left after the deletion.

7) When the number of consecutive bit-0s (*i.e.*, *non-skyrmions*) is too large or the shift distance is too long, specific bit errors might present [9].

3. Redesigning Data Structures for Racetrack Memories

3.1. Overview

This section discusses how existing algorithms and data structures can be redesigned to fit the characteristics of SK-RMs, so as to enhance the access performance, energy efficiency, and storage reliability. In particular, we present four architecture-level design ideas, whose objectives are:

- Reduce unnecessary shift operations to reduce the performance overheads.
- Avoid long-distance shifts to reduce the position errors.
- Support special-purpose data access operations such as range queries or historical queries.
- Minimize the creation (or destruction, equivalently) of skyrmions to save time and energy.

3.2. Pointer Usage Minimization

Linked data structures are everywhere. Popular examples of linked data structures, such as search trees and hash tables with chaining for collision resolution, have received much research attention [10], and their design spaces have almost been thoroughly explored under the RAM model. However, there are still missing pieces in the puzzle, when SK-RMs are adopted to store the data structure. This is because that the optimal *data placement policies* are not yet known for SK-RMs in general cases. Specifically, the two levels of data placement policies, namely the *microscopic* data placement policies (which determines how the different data bits of the same primitive variable are placed on the nanotracks) and *macroscopic* data placement policies (which arbitrate how different variables are organized on the multi-nanotrack architecture of SK-RMs), shall be smartly designed to achieve the design objectives. Only when the data placement policies are determined can the existing linked data structures be redesigned for SK-RMs.

Many existing linked data structures suffer the serious performance problem of *pointer chasing* [15] on the RM model, which can also be found on the RAM model, albeit less critical. To chase the pointers and walk the linked data structures, the memory access footprints tend to exhibit *random-hopping patterns*, which reduces the spatial locality of memory references, reduces the cache/buffer hit ratios, and degrades the performance of SK-RM, due to the shift overheads. In contrast, many array-based data structures often sequentially traverse the data items one by one, which incurs only the theoretically minimal shift overheads (that is, one shift operation per bit accessed). Thus, if possible, *data structures designed for SK-RM should limit the usage of pointers*. However, reducing the pointer chasing might increase the expected number of data bits that need to be traversed to access the demanded data. Space-efficient data structures such as *compact*, *succinct*, or *implicit* data structures [16] that less rely on the pointers might provide better performance on SK-RM, if with proper modifications.

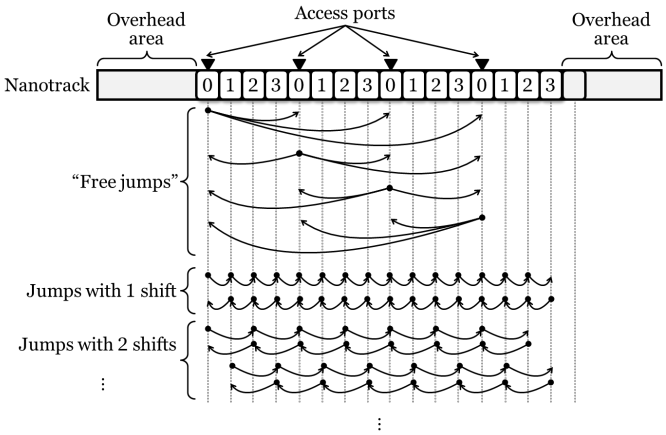

Figure 1. Jumps of different distances on SK-RMs.

3.3. Access Pattern Aware Data Placement [17]

To reduce the performance overheads due to the time-costly shift operations, a common idea might be to place the data that are expected to be accessed together at adjacent memory locations. Therefore, several existing work [17] attempt to statically analyze the data access patterns of applications, and determine the optimal data layout to minimize the shift overheads. However, difficulties might be encountered during the program analysis:

1) It is sometimes difficult or impossible to statically determine the data access pattern. If source code is unavailable, or the data access patterns of the target applications are highly dynamic, such analysis might be infeasible.
2) There is often a plural of access ports on each nanotrack of the SK-RM. Thus, using another access port to access the data bit just aligned with that access port can be done without any shift operation. From this perspective, the data bits stored on a nanotrack can be regarded as in a plural of *groups*, where we can perform random accesses on arbitrary data bits in the same group without incurring any shift operations (referred to as *free jumps* in Figure 1). Besides, how to minimize unnecessary switching of groups, or to avoid group switching with high shift overheads (referred to as *jumps with k shifts* with a large k), is also important. Concluded, the placement policies of data on SK-RMs should consider the intra-nanotrack level parallelism of data accesses, so as to alleviate the shift overheads.
3) A SK-RM chip has a large number of nanotracks, where the access ports on different nanotracks can be accessed

in parallel, without incurring shift overheads. The data placement policy should exploit the inter-nanotrack access parallelism to improve the throughput of data accesses.

3.4. Recursive Back-to-back Ordering of Data [8]

To address the performance problem due to pointer chasing of linked data structures [15], array-based data structures might be more suitable for SK-RMs. However, an issue with array-based data structures is its lacking of supports for efficient search, insert, or delete operations. Fortunately, in many application scenarios, data items or keys are not accessed alone. Instead, multiple data items or keys are often accessed together, and the data items or keys that are accessed together are often relevant to each other. For instance, *key-adjacent* or *time-adjacent* keys are often inserted, queried, or deleted together through *bulk insert operations, range query operations, or purge operations*. Thus, it is often useful to have a small SRAM or DRAM buffer to temporarily store the recently inserted data items, and, once the buffer gets full, its data items are sorted according to the timestamps, keys, or other fields, before actually flushed to the SK-RM. After the data items are flushed to the SK-RM, the DRAM buffer is emptied and can then accept new incoming data items. In this way, the data items in the SK-RM will form many *runs* (each of the same size as the DRAM buffer), whose keys are sorted. To manage the data items in the runs, the data can be further organized into levels, just like the popular *log-structured merge-tree (LSM-tree)*, which is widely adopted in key–value stores [18]. The major advantage to sort the data items in each run is the accelerated performance of range query operations, since data items whose keys are close to each other are guaranteed to be stored in adjacent memory locations in the SK-RM space.

3.5. Topology Regulation of Data Structures

Besides the performance problem due to shift operations, SK-RMs might suffer from specific flavors of data errors, which considerably degrade the reliability of data. For example, long-distance shifts are imprecise, and the to-be-accessed data might not be correctly aligned with the access port. As a result, the incorrect data bits might be read or written, resulting in numerous bit errors. To prevent from this, a possible idea is to make the data structures have *regularized topology* so that the data structure can be partitioned into multiple equal-sized parts. Each part should have its size being an exact multiple of the inter-port separation d (unit: bits), and align with an access port. As a result, the jumps between adjacent accesses to different parts do not generate any shift operations. For example, suppose that the bit-interleaved and block cluster architecture [14] are employed and $d = 8$, a *2-3-4 tree* would be more suitable than the *red–black tree* [10], since the red–black tree can potentially generate more shifts than the 2-3-4 tree because some red–black tree nodes might not be aligned with an access port of the nanotrack. Meanwhile, storing all keys of a 2-3-4 tree node in the beginning of the node might help reduce the shift operations that are required to determine the child link that should be followed.

4. Conclusion and Future Work

In this work, we revisit the computation models of SK-RMs, and, based on the obtained *racetrack memory model (RM model)*, further discuss potential design ideas to make existing data structures and algorithms friendly on SK-RMs. We hope that this work can shed light to future research work on the architectural design and management of SK-RMs, which is a promising choice of the memory–storage media for next-generation computing systems.

References

[1] I. Oukid and L. Lersch, "On the diversity of memory and storage technologies," *Datenbank-Spektrum*, vol. 18, no. 2, p. 121–127, Jun 2018. [Online]. Available: http://dx.doi.org/10.1007/s13222-018-0287-8

[2] S. Baek, H. G. Lee, C. Nicopoulos, and J. Kim, "A dual-phase compression mechanism for hybrid dram/pcm main memory architectures," in *Proceedings of the Great lakes symposium on VLSI*, 2012, pp. 345–350.

[3] Z. Sun, H. Li, Y. Chen, and X. Wang, "Variation tolerant sensing scheme of spin-transfer torque memory for yield improvement," in *2010 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2010, pp. 432–437.

[4] F. Clermidy, N. Jovanovic, S. Onkaraiah, H. Oucheikh, O. Thomas, O. Turkyilmaz, E. Vianello, J.-M. Portal, and M. Bocquet, "Resistive memories: Which applications?" in *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2014, pp. 1–6.

[5] W. Kang, X. Chen, D. Zhu, X. Zhang, Y. Zhou, K. Qiu, Y. Zhang, and W. Zhao, "A comparative study on racetrack memories: Domain wall vs. skyrmion," in *IEEE NVMSA '18*. IEEE, 2018, pp. 7–12.

[6] S. S. Parkin *et al.*, "Magnetic domain-wall racetrack memory," *Science*, vol. 320, no. 5873, pp. 190–194, 2008.

[7] X. Zhang, G. Zhao, H. Fangohr, J. P. Liu, W. Xia, J. Xia, and F. Morvan, "Skyrmion-skyrmion and skyrmion-edge repulsions in skyrmion-based racetrack memory," *Scientific reports*, vol. 5, p. 7643, 2015.

[8] Yun-Shan Hsieh et al., "Shift-limited sort: Optimizing sorting performance on skyrmion memory based systems," *Accepted and to Appear in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

[9] C. Zhang *et al.*, "Hi-fi playback: Tolerating position errors in shift operations of racetrack memory," in *ACM SIGARCH Computer Architecture News*, vol. 43, no. 3. ACM, 2015, pp. 694–706.

[10] T. H. Cormen *et al.*, *Introduction to Algorithms, Third Edition*, 3rd ed. The MIT Press, 2009.

[11] X.-Y. Hu, E. Eleftheriou, R. Haas, I. Iliadis, and R. Pletka, "Write amplification analysis in flash-based solid state drives," in *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, 2009, pp. 1–9.

[12] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. Elsevier, 2011.

[13] C. Lameter, "NUMA (non-uniform memory access): An overview," *Queue*, vol. 11, no. 7, pp. 40–51, 2013.

[14] R. Venkatesan *et al.*, "TapeCache: A high density, energy efficient cache based on domain wall memory," in *ACM/IEEE ISLPED '12*, 2012, pp. 185–190.

[15] K. Hsieh, S. Khan, N. Vijaykumar, K. K. Chang, A. Boroumand, S. Ghose, and O. Mutlu, "Accelerating pointer chasing in 3D-stacked memory: Challenges, mechanisms, evaluation," in *2016 IEEE 34th International Conference on Computer Design (ICCD)*. IEEE, 2016, pp. 25–32.

[16] G. Navarro and K. Sadakane, "Fully functional static and dynamic succinct trees," *ACM Transactions on Algorithms (TALG)*, vol. 10, no. 3, pp. 1–39, 2014.

[17] S. Gu *et al.*, "Area and performance co-optimization for domain wall memory in application-specific embedded systems," in *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 2015, p. 20.

[18] C. Luo and M. J. Carey, "LSM-based storage techniques: a survey," *The VLDB Journal*, vol. 29, no. 1, pp. 393–418, 2020.