ZUMA: Enabling Direct Insertion/Deletion Operations with Emerging Skyrmion Racetrack Memory

Zheng Liang, Guangyu Sun
Peking University
{liangzheng,gsun}@pku.com

Wang Kang, Xing Chen, Weisheng Zhao
Beihang University
{wang.kang,xing.chen0118,weisheng.zhao}@buaa.edu.cn

Abstract

Data insertion and deletion are common operations exist in various applications. However, traditional memory architecture can only perform an indirect insertion/deletion with multiple data read and write operations, which is significantly time and energy consuming. To mitigate this problem, we propose to leverage the unique capability of emerging skyrmion racetrack memory technology that it can naturally support direct insertion/deletion operations inside a racetrack. In this work, we first present a circuit level model for skyrmion racetrack memory. Then, we further propose a novel memory architecture to enable an efficient large size data insertion/deletion. With the help of the model and the architecture, we study several potential applications to leverage the insertion and deletion operations. Experimental results demonstrate that the efficiency of these operations can be substantially improved.

1 Introduction

Data insertion and deletion are common operations that can be employed in various levels of a computer system. Obviously, these operations are critical for those data structures (e.g., binary search tree), algorithms (e.g., insertion sort), and applications (e.g., databases), which want to maintain data in order. In addition, these operations can be leveraged to improve memory utilization in scenarios such as data allocation and garbage collections. Unfortunately, the memory architecture of a modern computer system is based on traditional random access memory technologies, such as SRAM and DRAM, which only support basic read and write operations to data stored in an array-like structure. Thus, an insertion or deletion operation is indirectly achieved using a lot of data read and write operations that induce intensive data movements. To this end, data management efficiency can be significantly improved if direct insertion and deletion operations are enabled.

The emerging skyrmion racetrack memory technology is a promising candidate to enable such attractive direct insertion and deletion operations in the future memory architecture design [27]. Skyrmion racetrack memory is a new generation of racetrack memory technology. It leverages magnetic skyrmions, which are particle-like spin textures moving along nanotrack driven by spin-polarized current, to store information. It has inherited advantages from the prior generation of domain-wall (DW) racetrack memory, which include high storage density and fast read/write speed [15]. Moreover, compared to traditional DW racetrack memory, skyrmion racetrack memory is more stable and requires less energy [8]. More importantly, it enables *a direct data insertion/deletion operation by injecting/evicting magnetic skyrmions to/from the nanotrack* [27].

Recently, traditional DW racetrack memory have been extensively studied as potential technology for on-chip cache architecture[20, 22], GPGPU register file[12], main memory, and even storage-class-memory[14]. However, there still lacks circuit level modeling and architecture level research on the new generation skyrmion racetrack memory technology. Especially, the attractive capability of enabling insertion and deletion operations are not exploited yet. To this end, we first provide a cross-layer modeling, design, and exploration to leverage this unique capability of skyrmion racetrack memory technology. Contributions of this work can be summarized as follows.

- We extend a traditional DW racetrack memory model to make it support skyrmion racetrack memory array, and the design space is explored considering different design parameters.
- We propose a novel memory architecture to enable an efficient large size data insertion, which can overcome the limitation constrained by the racetrack length.
- We study several potential applications, including cache tag and sorting algorithm, to leverage direct insertion/deletion operations enabled by skyrmion racetrack memory.
- We provide comprehensive experimental results to quantitatively evaluate the benefits of using skyrmion racetrack memory in different applications.

The rest of this work is organized as follows. In Section 2, we review the background of skyrmion racetrack memory technology. Circuit-level modeling of skyrmion racetrack memory is introduced in Section 3. We propose a relay architecture to support large data insertion/deletion in Section 4. Two potential applications are evaluated in Section 5 as case studies to leverage direct insertion/deletion, followed by a conclusion in the last section.

2 Preliminary

In this section, we review the basics of skyrmion racetrack memory technology. Related modeling work about prior generation DW racetrack memory technology is also introduced.

ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of a national government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.
DAC '19, June 2–6, 2019, Las Vegas, NV, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6725-7/19/06…$15.00
https://doi.org/10.1145/3316781.3317937

2.1 Basics of Skyrmion Racetrack Memory

Magnetic skyrmions are swirling topological configurations, which are mostly induced by chiral interactions between atomic spins in non-centrosymmetric magnetic bulks or in thin films with broken inversion symmetry. Owing to their intrinsic properties in nanoscale size, extremely low depinning current densities, high motion velocity and topological nontrivial feature [5, 8], they hold promise as information carriers in future ultra-dense, low-power memory, and logic devices. Furthermore, the standby energy consumption and heat generation during the processing and transportation of information can be efficiently reduced thanks to the merit of non-volatility. To date, advances have been made in the identification [7], creation/annihilation [11, 16], motion [11] of skyrmions at room temperature, which are the prerequisites to employ skyrmions in practical applications.

One of the most potential applications of skyrmions is to build racetrack memory [15]. Similar to the DW racetrack memory, skyrmion-based racetrack memory store data information by a sequence of skyrmion. It is expected to achieve higher package density and lower power consumption in comparison with those of DW racetrack memory because of the smaller size and lower depinning current density [16]. More importantly, by exploiting the unique properties of skyrmions, such as topological stability and particle-like behavior, it enables the new functionality of insertion and deletion [2, 27].

As shown in Figure 1a and 1b, a new skyrmion can be generated at the access port and then inserted into the racetrack with the help of injection current. All skyrmions on the right side of the access port are shifted one slot to the right. This is an illustration of data insertion for skyrmion racetrack memory. The deletion is an inverse process, in which a skyrmion is evicted followed by a partial shift. Obviously, the insertion and deletion "direction" can be controlled by the direction of injection current.

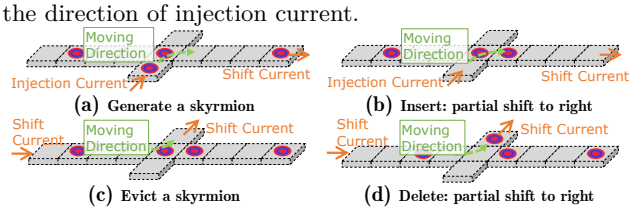


Figure 1: Illustration of Skyrmion Insertion and Deletion

2.2 DW Racetrack Memory Modeling

Prior work has provided detailed circuit level model [25] for domain wall racetrack memory based on a popular emerging memory modeling framework, NVSim [4]. NVSim is widely used to perform system-level exploration of emerging memory before real chip fabrication. Device level parameters, including cell layout, latency, energy, etc., are fed into NVSim framework. Then, it optimizes the NVM circuit designs, and evaluates the area, performance, and energy under given design constraints.

Compared to other emerging memory technologies (e.g. STT-RAM and PCM), one unique feature of racetrack memory is its tape-like cell shape, which make the circuit-level layout more complicated. To mitigate this problem, Macro Unit

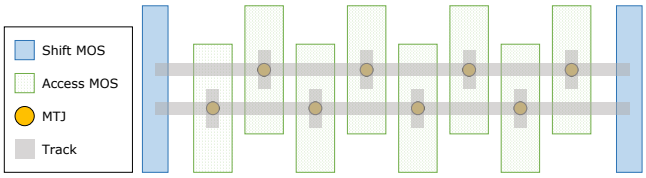


Figure 2: Top-view of a Macro Unit.

is proposed as the basic building block of DW racetrack memory in prior work [25]. A Macro Unit is normally composed of multiple cells of racetrack memory. The these tape-like cells are carefully organized inside a macro unit to optimize area efficiency. There are several important configuration parameters related to a macro unit and corresponding racetrack memory cell size. These parameters include cell numbers, port numbers, width/length of a racetrack, width/length of the access transistors, etc. In the next section, we will review these concepts with our detailed circuit-level modeling for skyrmion racetrack memory.

3 Circuit Design and Modeling

In this section, we first provide the macro unit design for skyrmion racetrack memory. Then, we introduce modeling for read, write, shift, and insertion/deletion operations. After that, a brief design space exploration is presented.

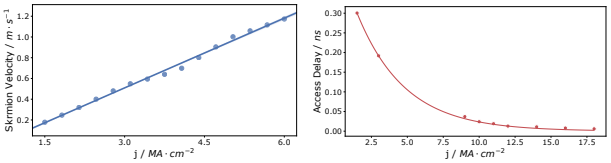
3.1 Macro Unit Design

A schematic top-view of one skyrmion racetrack Macro Unit is shown in Figure 2. It contains two skyrmion racetrack memory cells. Each cell has four access ports. Several important parameters and typical values of a macro unit are listed in Table 1. Unit F is the technology feature size. Using a specific skyrmion racetrack cell, the layout of a macro unit is determined by following design parameters: (1)L_{tr}, the effective storage length of a track, equal to bit numbers in a track, (2)PN, the number of access ports on a track, and (3)OD, overlapping degree, the number of parallel tracks that can be overlapped.

Parameter	Description	Typical Value
	Process Dependent Parameters	
W_{tr}	Width of track	$1F$
T_{tr}	Thickness of track	$0.4nm$
W_{MOS}	Gate width of MOS	$4F$
L_{MOS}	Length of MOS	$10F$
G_{MOS}	Minimum gap distance between MOSFETs	$1F$
ρ	Resistivity of track	$2.1\mu\Omega \cdot m$
j_{depin}	Depinning current	$0.7MA \cdot cm^{-2}$
	Design Parameters	
L_{tr}	Length of track	$16F$ - $2048F$
PN	Port number	depends on L_{tr}
OD	Overlapping degree	depends on L_{tr}, PN

Table 1: Macro Unit Parameters

Prior research has addressed that the major performance challenge of racetrack-like devices is its high latency of a shift operation [19, 21–26] to align data with access ports before read/write. In addition, extra overhead region reduces effective storage density. One method to alleviate both problems is to use multiple access ports, dividing the track into



(a) Shifting Speed (b) Generating latency

Figure 3: (a) Shifting speed and (b) skyrmion generating latency vs inject current density.

different sub-tracks. However, adding more ports may induce area overhead for access transistors. This is exploited in subsection 3.2. Having the basic macro unit, the data operations are modeled in the next subsection.

3.2 Data Access Operation Modeling

Read and Write Operations. These two operations are modeled similar to prior work [25], which are treated like STT-RAM operations. Read is performed via access port by measuring the magnetoresistance of the MTJ. Write is performed by activating a high write current pulse through MTJ, to change the magnetization direction of a skyrmion in the racetrack.

Shift Operations. Skyrmion motion along a nanotrack can be achieved by an electrical current through either the spin transfer torque (STT) or spin Hall effect (SHE) [5]. In this paper, we employ the case of utilizing the STT effect, the spin-polarized current is injected in-plane along the nanotrack. The skyrmion motion velocity can be calculated with the following equation,

$$v = \frac{\beta}{\alpha} v_s = -\frac{\beta}{\alpha} \frac{pa^3}{2eM_s} J_{STT} \tag{1}$$

where v_s is the velocity of the conduction electrons, which can be identified as the driving current density J_{STT} with a factor of $pa^3/2eM_S$. Here, p is the spin polarization of the electrical current, and e is the elementary charge. β is the coefficient of the nonadiabatic torque for the STT effect. M_S is the saturation magnetization. Typical values of these parameters can be found in related literature [2, 8, 27]. Figure 3a illustrates the relationship between shift speed and current density. Skyrmion moves faster under larger driven current density.

Insertion Operations. Each insertion can be decomposed of 2 stages: (1) generating one skyrmion at the injection side of the access port and (2) applying a driven current at the access port and corresponding end point of track to shift the inserted bit. Skyrmion creation via an electrical current or field is the most commonly studied method [5]. The procedure is considered as utilizing an electrical current to inject into a thin film [2, 8, 27]. Similar to prior work, we utilize micromagnetic simulation tool to simulate the skyrmion generation process. The latency of generating skyrmion under different current density is shown in Figure 3b. From the figure, we can tell that the shift operation is dominating in an insertion operation. Note that the driving current density is higher in such a partial shift for insertion, compared to that of shifting from one end of the racetrack. We can use Eq. 1 to calculate latency. Note that deletion is the inverse process of the insertion and the results are not shown due to page limit.

3.3 Design Space Exploration

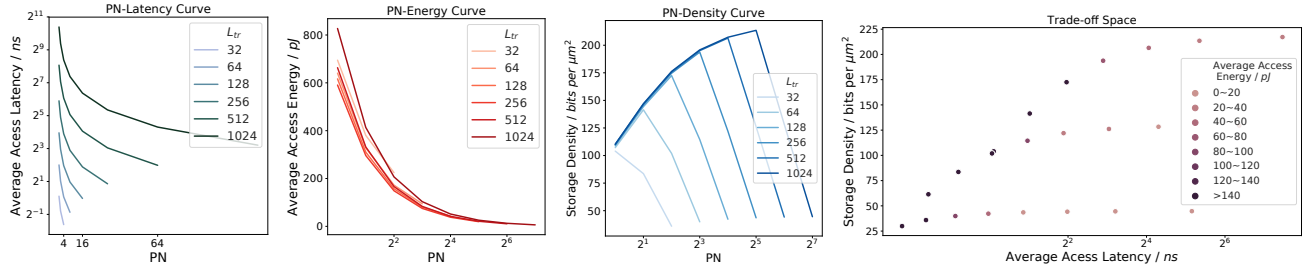
Having the circuit model, we perform a brief design space exploration to demonstrate the effect of two cortical parameters, L_{tr} and PN, on latency, energy, and density. We assume the highest overlapping degree is used by default. A fixed power supply voltage is provided and the current density is higher than the depinning current for each configuration. The memory size is set as 8MB.

As shown in Figure 4a, shorter tracks with more ports have lower access latency, thanks to higher driving current density and lower moving distance. While Figure 4b indicates that devices with more ports consume less energy due to shorter data movement. With regard to storage density, increasing PN may improve it at first by shrinking the size of the overhead region. However, this benefit diminishes with PN and the maximum OD available drops quickly. Thus, the storage density will drop quickly when PN exceeds a certain value. Longer tracks mean fewer macro units in an array, reducing area and power consumption of peripheral circuits. We plotted Pareto optimal points of our design in Figure 4d. The brightness of data points indicates average energy consumption. As shown in Figure 4d, with moderate latency requirements, our design can achieve considerable storage density and fairly high energy efficiency.

4 Relay Architecture for Large Data Insertion

Due to the depinning current density requirement, the length of a single racetrack is limited (normally less than 2048, as shown in Table 1). Thus, if the insertion region (memory region to be inserted) is larger than the track size, data will be shift out from one end of the racetrack and get lost. To overcome this problem, we propose a novel relay architecture to enable a larger insertion range. The basic idea is to read out the data before its being shifted out at the end of a racetrack. Then, the data is inserted into a neighbor racetrack. The pivot is to identify the location of this neighbour racetrack to support such a cross racetrack insertion.

An example is shown in Figure 6. As addressed before, a macro unit (MU) is the basic building block of skyrmion racetrack memory. All racetracks inside a macro unit are shifted or inserted/deleted together. To support cross macro unit data insertion, we add a sets of transmission gates between two adjacent macro units to mitigate data from one macro unit to the other. The detailed design is shown in Figure 5 (top). One extra access port is added at each end of a racetrack. As shown in the figure, the access port at right end of a racetrack in macro unit 3 is connected with the left end access end of a corresponding racetrack in macro unit 4. With the help of these extra ports, we can read out the data before its being shifted out from the end and write it to the neighbour racetrack. The next issue is to manipulate the transmission gates so that a proper insertion range is controlled.



(a) Access Latency (b) Access Energy (c) Storage Density (d) Design space for 8MB memory

Figure 4: Design space exploration.

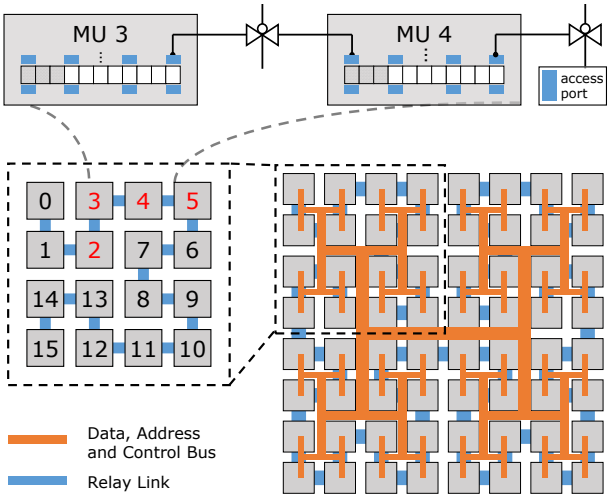


Figure 5: Detailed view of relay architecture.

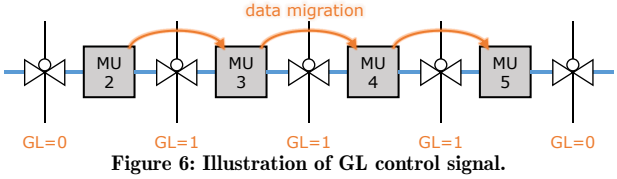


Figure 6: Illustration of GL control signal.

Assume that we want to group MU2-MU5 together to increase the insertion range by four times. This is a common case that a continuous region of memory is allocated to the user to support large size insertion. To achieve this, the transmission gates among these macro units are enabled with a control signal called GL, as shown in Figure 6. To achieve an efficient signal control, we provide a Hilbert-curve organization, as shown in Figure 5 (bottom). The index (address) of each macro unit is shown in the figure. A Hilbert curve is a continuous fractal space-filling curve. Given a 2D $2^k \times 2^k$ grid array, a Hilbert curve can be constructed by dividing the array into quad sub-arrays, recursively. Sub-arrays are linked in a U-shape topology. A $2^k \times 2^{k+1}$ grid array can be implemented by connecting 2 square arrays back to back.

After using the relay architecture, we can enable large insertion region across multiple macro units. The extra design overhead is moderate with extra transfer gates and one extra control line for GL. The energy consumption overhead is discussed as follows. If we involve N continuous macro units in the insertion, we will induce N extra reads and N extra writes of inserted data. In addition, the shift length is approximately increased by N times. A back-of-the-envelope calculation estimation is that the energy consumption is increased by approximately N times. This is reasonable as the process is equal to insert data into an N times length racetrack. The timing overhead can be hidden using a pipelining style, as long as these macro units can be shifted in parallel under the power-supply constraint. Otherwise, the process has to be performed in multiple steps.

We will justify this design in detail in our further work. A brief explanation is as follows. Plain X-Y order topology will introduce extremely long relay wires (blue lines in Figure5) between MUs in different rows (e.g., the last MU in the 1^{st} row and the first MU in the 2^{nd} row), while our Hilbert curve design can keep all relay wires local, reducing latency as well as saving energy. There also exist other designs like the s-shape curve, which can keep relay wires local. However, we don't have any good solution to decode the address mapping of MUs for such designs. Hilbert curve can be constructed recursively like traditional H-tree. Thus we only need to change the physical mapping of the decoders in the H-tree to serve our design. The control signal GL is also generated recursively by dividing and conquering the whole linear memory space.

5 Case Study and Evaluation

In this section, we will introduce how to leverage direct insertion/deletion operations in two applications: cache LRU replacement and insertion sorting. In this paper, we only demonstrate simple embedded applications, where there is no need for virtual memory or dynamic memory allocation, while far more complex applications can be applied too but with much more engineering efforts.

5.1 Cache LRU Replacement

LRU is an efficient replacement policy for set-associative caches. However, due to its high maintenance overhead [17], a pseudo-LRU rather than an original LRU policy is employed in high associativity caches. Fortunately, with the help of insertion/deletion enabled by skyrmion racetrack memory, the overhead of implementing a real LRU policy can be quite efficient. We provide a straightforward potential design in this subsection. Note that it can be further optimized in practice.

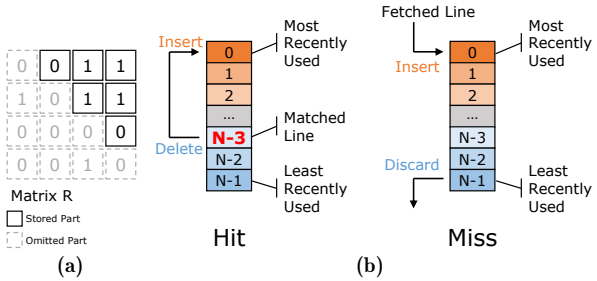


Figure 7: (a) SRAM LRU vs (b) Skyrmion racetrack LRU.

Given an N-way associative cache, a traditional implementation (shown in Figure 7a) is to use the upper triangular part of an $N \times N$ matrix R without diagonal to track the least recently used way per set, costing $N\left(N-1\right)/2$ bits [17] per set. Matrix R is initialized as 0. Each time way i is referenced, all bits in row i of R are set to 1 and then bits in column i are reset to 0. The least recently used way j is the one for which the entire row j is 0 (for bits in that row; a row can be empty) or the entire column j is 1 (for bits in that column; a column can be empty [17]).

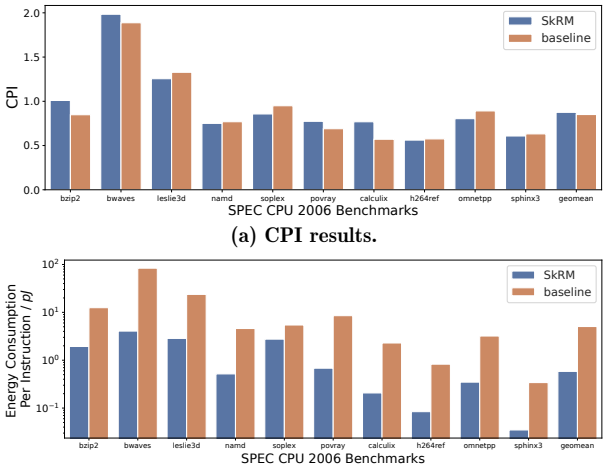
For our skyrmion racetrack, the update of tags is simple. First, we take one set of N-associativity cache as an example, as shown in the Figure 7b. Each time we get a hit, we delete corresponding cache line LRU index from the tag array, and add it to the head. If we encounter a miss, just add the LRU index of the cache line fetched to the head, squeezing out the least recently used one automatically.

We use gem5[1] integrated with our model for cycle accurate evaluation. We use the Alpha 21364[9, 13] processor as our baseline. We replace the L2 cache with skyrmion racetrack memory and compare iso-capacity performance. The system configuration of baseline is listed in Table 2. Architectural results are obtained by gem5. Power consumption and area are estimated by our model. Our benchmarks are gem5 supported items of SPEC CPU 2006 suite[18]. To reduce experiment time, we fast-forward 25% of total instructions to skip the beginning part and perform detailed simulations for 500 million instructions of each benchmark.

For fair comparison, we evaluate skyrmion racetrack based cache and the original SRAM based cache of Alpha 21364 both under 45nm process technologies. L2 cache is 32-way associative. Timing parameters of other parts in the Alpha 21364 processor under 45nm process is re-evaluated by FabScalar [3], a tool generating synthesizable superscalar processor designs.

Processor Core	Frequency	3.2 GHz
	Fetch Width	4
	Dispatch Width	4
	Issue Width	6
	Function Units	INT: 4, FP: 2
L1 I-/D-Cache	Size, Assoc, Lat.	64KiB, 8 ,1
L2 Cache	Size, Assoc, Lat.	2MiB, 32, 17
Cache Line Size	64 Byte	
System Memory	DDR3 1600 8x8	1 GiB

Table 2: System Configurations



(a) CPI results.

(b) Energy consumption results

Figure 8: Cache evaluation results for SPEC benchmarks.

For cache application, our latency-optimized configuration for skyrmion racetrack memory is $\{L_{tr} : 16, PN : 1\}$. Average shift latency of the skyrmion racetrack cache is 3.44 ns. Area of the skyrmion racetrack based cache is only 0.27 mm^2. The counterpart SRAM cache area is about 14.01 mm^2. Because of direct insertion/deletion, the area of LRU design is reduced by about **50x** after using skyrmion racetrack memory.

CPI results for these benchmarks are compared in Figure 8a. The performance is almost kept the same after the skyrmion racetrack cache. The shift latency is not an issue of performance. Although our skyrmion racetrack memory has a lower access latency, our LRU implementation has to shift to position 0 after each reference, doubling the access time. The results for energy consumption are listed in Figure 8b. It is easy to tell that energy consumption is reduced significantly after using skyrmion racetrack cache. Furthermore, higher associativity will widen the lead of our LRU cache design while a highly-associated LRU cache based on SRAM is even not feasible.

5.2 Insertion Sort

It is well known that comparison based sort algorithms require at least $\Omega\left(N \log N\right)$ comparisons[10]. Our solution for insertion sort algorithm is very straightforward. We keep an *ordered list* and insert data into it. To find a appropriate position to insert, we use binary search, which at most takes $\lfloor \log_2 N \rfloor$ comparisons. After inserting N elements, the sort process will complete. In the worst case, it will takes $\Sigma \lfloor \log_2 i \rfloor$ comparisons and $\Sigma\left(\lfloor \log_2 i \rfloor + 1\right)$ memory references. Thus, the time complexity of our solution is $\mathcal{O}\left(n \log n\right)$, achieving the asymptotic optimal.

Note that some traditional RAM-based algorithms can achieve the same asymptotic time complexity. In this part, we select a large scale input data (sort 50000 vectors by length) of the qsort benchmark in MiBench [6] suite as our benchmark. Each vector contains three 64-bit elements. We evaluate three solutions of sort algorithms using gem5 simulation and list them in Table 3. System configurations are same as those in Table 2 except that 2MB L2 cache is replaced

with 8MB scratchpad memory using SRAM or skyrmion racetrack memory. Skyrmion racetrack macro unit parameters are: $\{L_{tr} : 512, PN : 32\}$. Average access latency and access energy (including shift) are 8.23ns and 21pJ, respectively. And its area is $0.53mm^2$. For SRAM, the latency is 4.6ns and the access energy is 81pJ, with an area of 27.42 mm^2. Evaluation results show that using skyrmion racetrack memory outperform other solutions. We can conclude that our direct insertion solution brings significant speedup as well as impressive energy saving.

Algorithm	Execution Time(ms)	Memory Access(M)	Normalized Energy
Insertion Sort (SRAM)	1.1×10^4	1.9×10^3	1.5×10^5
Quick Sort (SRAM)	7.1×10^1	1.1×10^1	9.3×10^2
Insertion Sort (SkRM)	1.6×10^1	1.9	4.0×10^1

Table 3: Evaluation comparison for sort benchmark

6 Conclusion

Skyrmion racetrack memory technology has potential to enable direct insertion and deletion operations. In order to exploit the benefit in a higher level, we propose a circuit level model by extending prior framework. With the help of the circuit model, we can observe that such direct insertion/deletion operations are quite efficient but are limited inside a single track. To overcome this problem, we further propose a relay architecture. It can provide a near-linear scalable solution of extending insertion/deletion operations to a longer range. After applying the skyrmion racetrack memory in two applications, evaluation results demonstrate that data management become more efficient with the help of direct insertion and deletion operations.

7 Acknowledgments

This work is supported by NSF China 61832020 and 61572045. The authors thank Dr. Chao Zhang from PKU and Dr. Shuo Li from NUDT for their guide on gem5.

References

[1] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The Gem5 Simulator. *SIGARCH Comput. Archit. News* (Aug 2011).

[2] Xing Chen, Wang Kang, Daoqian Zhu, Xichao Zhang, Na Lei, Youguang Zhang, Yan Zhou, and Weisheng Zhao. 2018. Complementary Skyrmion Racetrack Memory Enables Voltage-Controlled Local Data Update Functionality. *IEEE Transactions on Electron Devices* (2018).

[3] N. K. Choudhary, S. V. Wadhavkar, T. A. Shah, H. Mayukh, J. Gandhi, B. H. Dwiel, S. Navada, H. H. Najaf-abadi, and E. Rotenberg. [n. d.]. FabScalar: Composing synthesizable RTL designs of arbitrary cores within a canonical superscalar template. In *2011 38th Annual International Symposium on Computer Architecture (ISCA)*.

[4] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi. 2012. NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31, 7 (July 2012).

[5] Albert Fert, Vincent Cros, and Joao Sampaio. 2013. Skyrmions on the track. *Nature nanotechnology* (2013).

[6] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. 2001. MiBench: A free, commercially representative embedded benchmark suite. In *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization.*

[7] Christian Hanneken, Fabian Otte, André Kubetzka, Bertrand Dupé, Niklas Romming, Kirsten Von Bergmann, Roland Wiesendanger, and Stefan Heinze. 2015. Electrical detection of magnetic skyrmions by tunnelling non-collinear magnetoresistance. *Nature nanotechnology* (2015).

[8] W. Kang, Y. Huang, X. Zhang, Y. Zhou, and W. Zhao. 2016. Skyrmion-Electronics: An Overview and Outlook. *Proc. IEEE* (2016).

[9] R. E. Kessler. 1999. The Alpha 21264 microprocessor. *IEEE Micro* (March 1999).

[10] Donald Ervin Knuth. 1997. *The art of computer programming: sorting and searching*. Vol. 3. Pearson Education.

[11] Wataru Koshibae and Naoto Nagaosa. 2014. Creation of skyrmions and antiskyrmions by local heating. *Nature communications* (2014).

[12] M. Mao, W. Wen, Y. Zhang, Y. Chen, and H. Li. 2014. Exploration of GPGPU Register File Architecture Using Domain-wall-shift-write based Racetrack Memory. In *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*.

[13] S. S. Mukherjee, P. Bannon, S. Lang, A. Spink, and D. Webb. 2001. The Alpha 21364 network architecture. In *HOT 9 Interconnects. Symposium on High Performance Interconnects*.

[14] E. Park, S. Yoo, S. Lee, and H. Li. 2014. Accelerating Graph Computation with Racetrack Memory and Pointer-Assisted Graph Representation. In *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*. 1–4.

[15] Stuart SP Parkin, Masamitsu Hayashi, and Luc Thomas. 2008. Magnetic Domain-Wall Racetrack Memory. *Science* (2008).

[16] João Sampaio, Vincent Cros, Stanislas Rohart, André Thiaville, and Albert Fert. 2013. Nucleation, stability and current-induced motion of isolated magnetic skyrmions in nanostructures. *Nature nanotechnology* 8, 11 (2013), 839.

[17] Alan Jay Smith. 1982. Cache Memories. *ACM Comput. Surv.* 14, 3 (Sept. 1982), 473–530.

[18] Cloyce D. Spradling. 2007. SPEC CPU2006 Benchmark Tools. *SIGARCH Comput. Archit. News* (Mar 2007).

[19] G. Sun, C. Zhang, H. Li, Y. Zhang, W. Zhang, Y. Gu, Y. Sun, J. . Klein, D. Ravelosona, Y. Liu, W. Zhao, and H. Yang. 2015. From device to system: Cross-layer design exploration of racetrack memory. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*.

[20] Z. Sun, X. Bi, A. K. Jones, and H. Li. 2014. Design exploration of racetrack lower-level caches. In *2014 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*.

[21] Zhenyu Sun, Wenqing Wu, and Hai (Helen) Li. 2013. Cross-layer Racetrack Memory Design for Ultra High Density and Low Power Consumption. In *Proceedings of the 50th Annual Design Automation Conference*.

[22] Rangharajan Venkatesan, Vivek Kozhikkottu, Charles Augustine, Arijit Raychowdhury, Kaushik Roy, and Anand Raghunathan. 2012. TapeCache: A High Density, Energy Efficient Cache Based on Domain Wall Memory. In *Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design*.

[23] R. Venkatesan, V. J. Kozhikkottu, M. Sharad, C. Augustine, A. Raychowdhury, K. Roy, and A. Raghunathan. 2016. Cache Design with Domain Wall Memory. *IEEE Trans. Comput.* (2016).

[24] Rangharajan Venkatesan, Mrigank Sharad, Kaushik Roy, and Anand Raghunathan. 2013. DWM-TAPESTRI - an Energy Efficient All-spin Cache Using Domain Wall Shift Based Writes. In *Proceedings of the Conference on Design, Automation and Test in Europe*.

[25] Chao Zhang, Guangyu Sun, Weiqi Zhang, Fan Mi, Hai Li, and W. Zhao. 2015. Quantitative modeling of racetrack memory, a tradeoff among area, performance, and power. In *The 20th Asia and South Pacific Design Automation Conference*.

[26] H. Zhang, C. Zhang, Q. Hu, C. Yang, and J. Shu. 2018. Performance Analysis on Structure of Racetrack Memory. In *2018 23rd Asia and South Pacific Design Automation Conference*.

[27] D. Zhu, W. Kang, S. Li, Y. Huang, X. Zhang, Y. Zhou, and W. Zhao. 2018. Skyrmion Racetrack Memory With Random Information Update/Deletion/Insertion. *IEEE Transactions on Electron Devices* 65, 1 (2018), 87–95.