RTSim: A Cycle-accurate Simulator for Racetrack Memories

Asif Ali Khan[1], Fazal Hameed[1], Robin Bläsing[2], Stuart Parkin[2] and Jeronimo Castrillon[1]
[1]Chair for Compiler Construction TU Dresden, Germany
[2]Max Planck Institute of Microstructure Physics Halle, Germany

Abstract—*Racetrack memories* (RTMs) have drawn considerable attention from computer architects of late. Owing to the ultra-high capacity and comparable access latency to SRAM, RTMs are promising candidates to revolutionize the memory subsystem. In order to evaluate their performance and suitability at various levels in the memory hierarchy, it is crucial to have RTM-specific simulation tools that accurately model their behavior and enable exhaustive design space exploration. To this end, we propose *RTSim*, an open source cycle-accurate memory simulator that enables performance evaluation of the domain-wall-based racetrack memories. The skyrmions-based RTMs can also be modeled with RTSim because they are architecturally similar to domain-wall-based RTMs. RTSim is developed in collaboration with physicists and computer scientists. It accurately models RTM-specific *shift operations*, access ports management and the sequence of memory commands beside handling the routine read/write operations. RTSim is built on top of NVMain2.0, offering larger design space for exploration.

Index Terms—Memory simulator, racetrack memory, domain wall memory, memory system, main memory, cache, scratchpad, simulation, emerging memory technologies, NVM.

1 Introduction

With the transition of computer systems from multi- to many-cores, the search for low-power and high-capacity memories has gathered unprecedented momentum. As a result, multiple *volatile* and *non-volatile memories* (NVMs) have emerged in the last decades. The evolutionary DRAM standards (low power DDR4, die-stacked WIO, HBM and HMC), *spin-transfer-torque RAM* (STT-RAM), *phase change memory* (PCM), *resistive RAM* (RRAM) and *racetrack memory* (RTM) are prominent examples. While some of these memories have already made it to the market, others are still in their infancy. Amongst all, the racetrack memory is believed to offer "faster-than-Moore's-law" scaling path and is a promising candidate to bridge the processor memory gap [10], [11].

Proper evaluation and exploration and of these new memory technologies require availability of accurate simulation tools. In the past, memory researchers have developed multiple device and architecture level memory simulators. In particular, DRAMSim [16], DRAMSim2 [14], DRAMSys [5] and Ramulator [7] are available to explore wide varieties of DRAM standards. Similarly, new memory simulators have been developed to model these emerging NVMs as well. For instance, NVMain [12], NVMain2.0 [13] and the recently extended NVMain [6] can model STT-RAM, PCM, HMC and WIDE I/O besides modeling the conventional DRAMs and SRAM technologies.

The relatively newer spin-orbitronics based RTMs are fundamentally different than all existing memory technologies. Unlike contemporary memory technologies, a single access point in RTMs can store multiple bits i.e., 1 to 100. These bits are stored in the form of *magnetic domains* in a tape-like structure called *track* which can be placed vertically (3D) or horizontally (2D) on the surface of a silicon wafer as depicted in Figure 1. Each track in RTM is equipped with one or more *magnetic tunnel junction* (MTJ) sensors, referred to as *access ports* (AP), that are used to perform read/write operations.

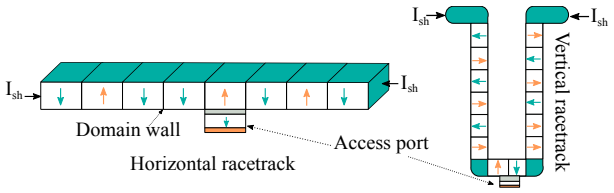

Fig. 1. Racetrack horizontal and vertical placements (I_{sh} represents the shift current)

To evaluate the performance of RTMs and enable system design, new simulation tools are needed that accurately model the shift operations and manage the access ports. In the literature, people have reported modifications to existing simulators such as gem5 [2], [8], simplescalar [1] and NVMain for exploring RTMs at various hierarchy levels in the memory subsystem [15], [19]. However, these extensions are not available in the public domain. This not only deprives the memory research community of exploring RTMs but also makes it near to impossible to compare results, a process that is key for advancing the field.

To fill this gap, we present RTSim; an architectural-level cycle-accurate simulation framework for RTMs that accurately models the shift operations, manages the access ports and the RTM specific memory commands sequence.

A.A. Khan, F. Hameed and J. Castrillon are with the Chair for Compiler Construction TU Dresden while R. Bläsing and S. Parkin are with Max Planck Institute of Microstructure Physics at Halle, Germany. F. Hameed is also affiliated with the Institute of Space Technology, Pakistan. E-mails: {asif_ali.khan, fazal.hameed, jeronimo.castrillon}@tu-dresden.de {blaesing, stuart.parkin}@mpi-halle.mpg.de

TABLE 1
RTM device level parameters [17]

Parameter	Value
Thickness, width and length of the nano-wire	6 nm, $1F$ and $128F$
Domain Length	$2F$
Nanowire resistivity	$4.8 \times 10^{-7} \Omega$ m
Critical current density for Shifting (J_c)	$6.2 \times 10^7 A/cm^2$
Critical current density for write (J_w)	$5.7 \times 10^6 A/cm^2$

RTSim is configurable and allows architects to explore the design space of RTMs by varying the design parameters such as the number of tracks, domains and access ports per track, port update policy and the domains access policy. The modular design of RTSim facilitates the development and easy integration of new extensions such as position error correction schemes [18].

2 RTSim Overview

RTSim is built on top of NVMain2.0. We have made necessary modifications to most of the simulator modules such as *address translators* and *memory controllers* to cater for RTMs. The modifications to the address translator are required to translate the physical address to the corresponding RTM device address which is different than the device addresses of other memory technologies. A bank in an RTM is made up of one or more subarrays which in turn consists of multiple *Domain Block Clusters* (DBCs) as shown in Fig. 2. Each DBC contains M tracks and N domains per track, where each domain stores a single bit. Accessing a bit from a track requires shifting and aligning the corresponding domain to the track's port position. Typically, an M-bit variable is distributed across M tracks of a DBC. The domains of all tracks in a particular DBC move in a lock step fashion so that all M bits of a variable are aligned to the port position at the same time for simultaneous access.

In some specific cases, storing a variable serially, in a single racetrack, may be more beneficial compared to the aforementioned distributed layout. RTSim implements both layouts and allows designers to set the *Layout* parameter to either *Interleaved* or *Serial* in the configuration file.

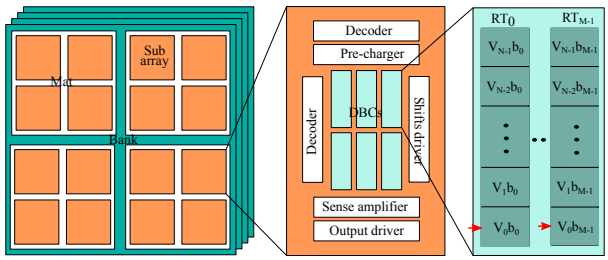

Fig. 2. Racetrack memory architecture (RT: track, b:bit)

As shown in Fig. 3, RTSim requires a configuration file and a memory request stream. The configuration file consists of the system as well as latency/energy parameters. The system parameters such as number of ranks, banks, DBCs and word size (number of racetracks per DBC) are independent of the RTM device and can be configured according to design requirements. The device level parameters are listed in Table 1. Using these parameters, the latency and energy values can be extracted from circuit simulators such as NVSim [3] or Destiny [9].

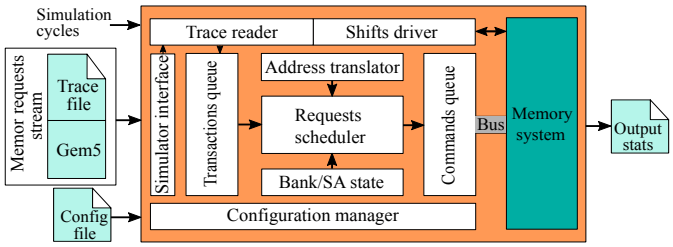

Fig. 3. RTSim overview

2.1 Address mapping scheme

RTSim translates the physical address of the CPU requests to the corresponding memory address. The memory address in RTSim consists of domain ID, DBC ID, subarray ID, bank ID, rank ID and channel ID. The lower $log_2 W$ bits correspond to the word bytes where W represents the word size in bits.

The address mapping scheme in RTMs is more crucial compared to other memories. This is due to the fact that other memories optimize address mapping for exploiting locality, minimizing bank conflicts and improving parallelism. The address mapping scheme in RTMs should also optimize the request stream for consecutive accesses in order to mitigate the number of shifts. This implies that spatially adjacent memory requests should be assigned to consecutive domains in the same DBC. The default RTSim addressing scheme looks like *RK:BK:CH:DBC:DOM* and can be configured as per the design objectives.

2.2 Memory controller

The memory controller in RTSim buffers CPU requests in a *transaction queue*. Subsequently, each transaction is converted into a set of RTM commands which are placed in a *command queue*. Similar to NVMain2.0, the model and size of command queues are configurable.The memory controller schedules and issues RTM commands to the memory banks in an out-of-order manner while respecting both timing and flow of commands constraints. Memory requests are reordered based on the current ports positions and commands are issued such that the shift overhead is minimized. Requests starvation is avoided with a set threshold.

Once a command is issued, respective sanity checks (for timing constraints) are performed at rank, bank and subarray levels and simulation statistics are updated. The shift statistics in RTSim are computed at the DBC level and can be accumulated at more abstract levels e.g. subarray, bank, rank, and channel levels. At completion, requests are returned to the memory controller which removes them from their respective queues and returns them to the owner of the request.

2.3 Access ports management

Since the number of access ports per track is much smaller than the number of domains, ports are always shared among domains. While increasing the sharing degree increases the area efficiency, it leads to an increased number of shifts which in turn increases the average access latency. RTSim faithfully models contention that arises due to shift delay, queuing delays and bank/port conflicts.

RTSim allows users to configure the number of ports. The memory controller maintains the status/positions of all access ports corresponding to each track. In the default interleaved data layout, tracks are grouped into DBCs (cf.

TABLE 2
RTM configuration parameters

Parameter	Description	value
DBCS	Number of DBCs per bank	Positive integer (depending on the memory size and configuration)
Domains	Number of domains per track	1-100 (default value is 64)
WordSize	Number of tracks per DBC	1 to N bits (default value is 32 bits)
nPorts	Number of access ports per track	Less than or equal to the number of domains (default is 1)
PortAccess	Port access policy	Static / dynamic (default is static)
PortsInitPos	Initial position of the access ports	Assigned automatically if not specified (default 0 for single port)
Layout	variable storage format	Serial / Interleaved
PortUpdate	Port position after each access	eager / lazy

Fig. 2) and all ports in a DBC move together in the lockstep fashion. This implies that the port positions of all tracks in a DBC are always the same. At abstract level, it appears as if the ports are per DBC and not per track inside the DBC. In the serial layout, ports of individual tracks are managed separately.

The memory controller also decides which port should access a certain domain if there is more than one access ports per track. The idea is similar to the *tape head selection* policy in [15] and is referred to as the *port access policy* in RTSim. Similar to other parameters, the port access policy in RTSim is *configurable*, and can be set as either *static* or *dynamic*. In the static port access policy, each domain is assigned an access port statically depending on its initial placement. For instance, if a track has N domains and P access ports, N/P domains are statically assigned to each access port. Each access port is then responsible for accessing its set of domains even if the desired domains are closer to the other access port(s).

On the contrary, in the dynamic port access policy, the closest access port accesses the requested domain. While the dynamic policy will tend to reduce the number of shifts compared to the static policy, it may increase the number of *overflow* bits. The overflow bits are required to prevent the loss of data and store the shifted domains beyond the shift ports. For a single port per track, N overflow bits are needed to store the shifted domains. For P access ports and static port policy, the amount of overflow bits reduces to N/P. In the dynamic case, P access ports still require N overflow bits.

RTSim supports two different port update policies. Following a memory access, the port positions in RTM are updated according to the *PortUpdate* parameter specified in the configuration file. In the default *lazy* policy, the port that accesses the current domain stays at the position of the current access and all other ports positions are updated accordingly. On the contrary, all port positions in the *eager* policy are restored to their initially assigned locations after each memory access. While the eager policy is easy to implement and simplifies ports selection, it may significantly increase the number of shifts. The configurability of RTSim allows designers to choose the best configuration by performing the aforementioned trade off analysis.

2.4 Latency and energy models

RTSim offers flat models for latency and energy. The latency/energy values are extracted from Destiny [9], employing device level parameters from our in-house physics lab. The latency and energy models in RTSim use these numbers along with the memory access statistics to compute the total latency and energy of the memory subsystem.

As an example, we simulate a 1 MB cache in Destiny to obtain the latency and energy numbers for read/write/shift operations in RTMs. These sample values are given in the RTSim configuration file *RTM.config*. Every time the simulator performs a memory operation, the energy and timing statistics are updated accordingly.

2.5 RTMs specific configuration parameters

Most of the configuration parameters in RTSim are similar to NVMain. The newly added RTM-specific parameters are described in Table 2. The initial positions of the access ports are set automatically if not specified in the configuration file.

RTSim, being developed on top of NVMain2.0, also supports other NVMs. The RTM-specific features are only enabled if the corresponding RTM configuration file is provided. The integration with NVMain2.0 facilitates interface to other simulators. For instance, the existing nvmain-gem5 patch can be employed to simulate RTMs in full system mode with the gem5 system simulator. In a stand-alone mode, memory traces are fed to RTSim to simulate an RTM-based memory subsystem.

3 Case Studies

This section presents a case study to validate the accuracy of the simulation framework. RTSim adopts the timing and energy models from NVMain2.0. Since these models are already verified with the industrial Verilog models and validated against other established memory simulators, we only focus on verifying the modeling of shift operations and the access ports management.

Unfortunately, no commercial/research prototypes are available for RTMs which can be used as a reference for validation. We work around this problem by establishing our own simulation target. We use synthetic memory traces as golden references for which we can predict the number of shifts. The memory traces are developed in a careful manner such that requests hop among domains, DBCs, subarrays and banks. We provide these traces to RTSim to verify the number of shifts. A sample memory trace with expected and observed number of shifts is shown in Fig. 5.

After verifying the functional correctness of RTSim, we stress-test it by running the whole set of SPEC2006 [4] benchmarks. The vertical axis (log scale) in Fig. 4 reports the number of shifts. The figure highlights the impact of varying the port access policy as well as the number of ports while using the best performing lazy ports update policy. As can be seen, increasing the number of ports reduces the number of shifts as expected. Similarly, the dynamic port access policy often reduces the number of shifts compared

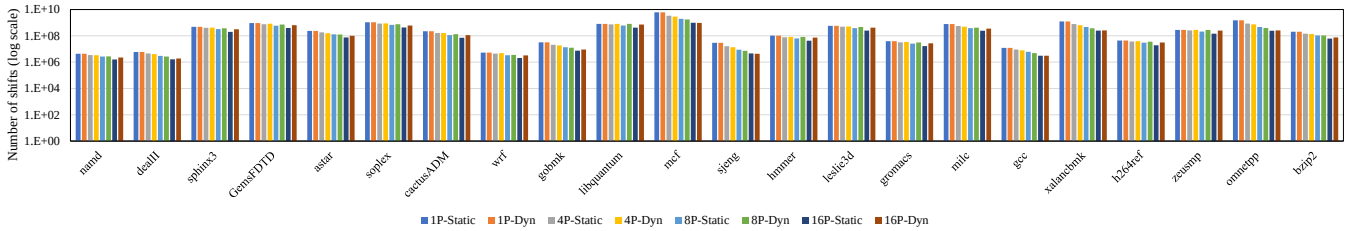

Fig. 4. Impact of varying number of access ports on the number of shifts in SPEC2006 benchmarks

Shifts per memory request

Req	Phy-address	Mem-address*	Num-shifts
R	0x1e0	0:0:0:0:0:15	15 x 32
W	0x86bc0	67:0:1:0:0:30	30 x 32
R	0x86bc0	67:0:1:0:0:30	0
R	0x86b00	83:0:1:0:0:24	24 x 32
R	0x87e38	75:0:3:0:0:49	49 x 32
R	0x3c0	0:0:0:0:0:30	15 x 32
R	0x1400	0:0:2:0:0:32	32 x 32
*DBC:RK:BK:CH:SA:DOM			5280

Config

```
nPorts 1
PortAccess static
BANKS 4
RANKS 1
CHANNELS 1
DBCS 128
DOMAINS 64
; WordSize in bits
WordSize 32
```

Snapshot from RTSim output

```
RTM.channel0.rank0.bank0.subarray0.totalnumShifts 960
RTM.channel0.rank0.bank1.subarray0.totalnumShifts 1728
RTM.channel0.rank0.bank2.subarray0.totalnumShifts 1024
RTM.channel0.rank0.rank0.bank3.subarray0.totalnumShifts 1568
```

Fig. 5. Number of shifts computed from the synthetic trace and reported by RTSim. The memory request types and physical addresses are taken from the trace file while the memory address is the output of the RTSim decoder. The *Num-shifts* are manually computed.

to the static port access policy. However, for 16 ports per track configuration, the static policy mostly outperforms the dynamic access policy. This is due to the fact that the worst-case shifts in the static policy are always 4 while in the dynamic policy this can increase up to 63. Detailed analysis of the two policies is beyond the scope of this paper. RTSim enables memory researchers to perform exptensive pros/cons evaluation of the two policies.

4 Conclusions

Racetrack memory is a promising alternative to existing (non-)volatile memories. The lack of simulation and exploration tools in the public domain hinders their expeditious development and exploration for novel memory subsystem. To overcome this, we present RTSim, a cycle-accurate simulation tool for racetrack memories. RTSim accurately models the shift operations and manages the access ports in RTMs, beside modeling the routine memory operations. The memory controller in RTSim ensures that commands are issued to memory in a proper order and all timing constraints are satisfied. We validate the shift model of RTSim with a set of synthetic memory traces and exemplarily show shift analysis for SPEC2006 benchmarks using different configurations. Being the first RTM simulator in the public-domain, we believe RTSim will alleviate the difficulties in RTMs design space exploration and become a useful tool for the community.

Acknowledgments

This work was partially funded by the German Research Council (DFG) through the Cluster of Excellence 'Center for Advancing Electronics Dresden' (cfaed).

References

[1] T. Austin, E. Larson, and D. Ernst. Simplescalar: an infrastructure for computer system modeling. *Computer*, 35(2):59–67, Feb 2002.

[2] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, Aug. 2011.

[3] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi. 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):994–1007, July 2012.

[4] J. L. Henning. Spec cpu2006 benchmark descriptions. *SIGARCH Comput. Archit. News*, 34(4):1–17, Sept. 2006.

[5] M. Jung, C. Weis, and N. Wehn. Dramsys: A flexible dram subsystem design space exploration framework. *IPSJ Transactions on System LSI Design Methodology*, 8:63–74, 2015.

[6] A. A. Khan, F. Hameed, and J. Castrillon. Nvmain extension for multi-level cache systems. In *Proceedings of the Rapido'18 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*, RAPIDO '18, pages 7:1–7:6, New York, NY, USA, 2018. ACM.

[7] Y. Kim, W. Yang, and O. Mutlu. Ramulator: A fast and extensible dram simulator. *IEEE Comput. Archit. Lett.*, 15(1):45–49, Jan. 2016.

[8] C. Menard, J. Castrillón, M. Jung, and N. Wehn. System simulation with gem5 and systemc the keystone for full interoperability. 2017.

[9] S. Mittal, R. Wang, and J. Vetter. Destiny: A comprehensive tool with 3d and multi-level cell memory modeling capability. *Journal of Low Power Electronics and Applications*, 7(3), 2017.

[10] S. Parkin, M. Hayashi, and L. Thomas. Magnetic Domain-Wall Racetrack Memory. 320:190–194, 05 2008.

[11] S. Parkin and S.-H. Yang. Memory on the Racetrack. 10:195–198, 03 2015.

[12] M. Poremba and Y. Xie. Nvmain: An architectural-level main memory simulator for emerging non-volatile memories. In *2012 IEEE Computer Society Annual Symposium on VLSI*, pages 392–397, Aug 2012.

[13] M. Poremba, T. Zhang, and Y. Xie. Nvmain 2.0: A user-friendly memory simulator to model (non-)volatile memory systems. *IEEE Computer Architecture Letters*, 14(2):140–143, July 2015.

[14] P. Rosenfeld, E. Cooper-Balis, and B. Jacob. Dramsim2: A cycle accurate memory system simulator. *IEEE Comput. Archit. Lett.*, 10(1):16–19, Jan. 2011.

[15] R. Venkatesan, V. Kozhikkottu, C. Augustine, A. Raychowdhury, K. Roy, and A. Raghunathan. 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*, ISLPED '12, pages 185–190, 2012.

[16] D. Wang, B. Ganesh, N. Tuaycharoen, K. Baynes, A. Jaleel, and B. Jacob. Dramsim: A memory system simulator. *SIGARCH Comput. Archit. News*, 33(4):100–107, Nov. 2005.

[17] C. Zhang, G. Sun, W. Zhang, F. Mi, H. Li, and W. Zhao. Quantitative modeling of racetrack memory, a tradeoff among area, performance, and power. In *The 20th Asia and South Pacific Design Automation Conference*, pages 100–105, Jan 2015.

[18] C. Zhang, G. Sun, X. Zhang, W. Zhang, W. Zhao, T. Wang, Y. Liang, Y. Liu, Y. Wang, and J. Shu. Hi-fi playback: Tolerating position errors in shift operations of racetrack memory. In *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, pages 694–706, June 2015.

[19] H. Zhang, C. Zhang, Q. Hu, C. Yang, and J. Shu. Performance analysis on structure of racetrack memory. In *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 367–374, Jan 2018.