ERMES: Efficient Racetrack Memory Emulation System based on FPGA

Fanny Spagnolo[†], Salim Ullah[‡], Pasquale Corsonello[†], Akash Kumar[‡]
[†]*Department of Informatics, Modelling, Electronics and Systems Engineering*, University of Calabria – 87036 Rende, Italy
[‡]*Chair of Processor Design, Center for Advancing Electronics Dresden*, Technische Universität Dresden – 01062 Dresden, Germany
Email: f.spagnolo@dimes.unical.it, salim.ullah@tu-dresden.de, p.corsonello@unical.it, akash.kumar@tu-dresden.de

***Abstract*—With the scaling of CMOS technology almost over, non-volatile memories based on emerging technologies are gaining considerable popularity. Particularly, spintronic-based Racetrack memories (RTMs) exhibit unprecedented storage capacity, as well as reduced energy per operation and high write endurance, which make them promising candidates to revolutionize the architecture of memory sub-systems. However, since RTM exploits shifting of magnetic domains to align the required data with the access port, its read/write latency is not constant. Due to this behaviour, several performance optimizations related to the target application may be introduced either on memory architecture or data placement or both. To this purpose, specific tools able to emulate the timing characteristics of RTMs are highly desired. Unfortunately, existing software-based simulators show poor flexibility and run-time. To address such limitations, this paper presents a new emulation system for RTMs based on heterogeneous FPGA-CPU Systems-on-Chips (SoCs). Thanks to its high flexibility, the proposed emulator can be easily configured to evaluate different memory architectures. In addition, the CPU can be used to stimulate the RTM architecture under test with appropriate benchmarks, thus providing a fast self-contained evaluation environment. As case study, ERMES has been implemented within the Xilinx Zynq Ultrascale XCUZ9EG SoC to evaluate performances of several memory configurations when running benchmark applications from the MiBench suite, experiencing a speed-up higher than $\times 146$ over software-based simulators.**

***Index Terms*—Racetrack Memories, FPGA-based emulator, Computer aided design techniques**

I. Introduction

In the era of the Internet-of-Things (IoT), memory-hungry applications, such as big data analytics, multimedia processing and machine learning algorithms, have become particularly popular [1]. As illustrated in Fig. 1, IoT edge platforms typically rely on embedded devices, with one or more computing components, such as Central Processing Unit (CPU), Graphical Processor Unit (GPU) and custom hardware accelerators. Each computing unit has its own internal memory bank, responsible for caching data and instructions that are accessed frequently. Furthermore, specific memory sub-systems are used to favour data sharing among different computing units and with other external memory peripherals.

In the above mentioned scenarios, designing memory sub-systems with large storage capacity and suitable to operate in energy-, area- and time-constrained environment has become quite challenging [2]. Technological limits shown by conventional SRAM and DRAM architectures have driven the research community towards new emerging technologies characterized by more favourable scalability factors and lower power dissipation. In the recent past, several non-volatile memories (NVMs) have emerged as promising alternatives, such as spin transfer torque RAMs (STT-RAMs) [3], phase change memories (PCMs) [4], resistive RAMs (ReRAMs) [5] and racetrack memories (RTMs) [6]. However, while on the one hand the above NVMs offer high energy efficiency, on the other hand most of them [3]- [5] suffer from limited endurance, high write time and large cell size, which restrict their applicability in embedded devices. Most representative memory architectures are compared in terms of performance, area and energy in Fig. 2. It can be observed that RTMs combine strengths of different technologies, providing lower energy per operation and the write endurance closer to traditional RAMs, a very low leakage dissipation and an unprecedented storage density capacity [7].

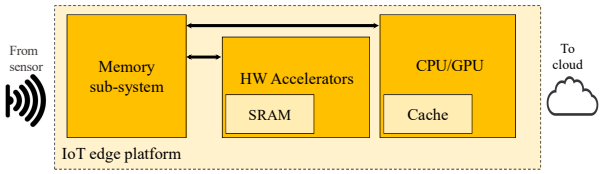

Fig. 1. Architecture of a typical IoT edge platform.

The information in RTM is stored within magnetic nanowires (or tracks), with the magnetic *domains* serving as bits (0 or 1 depending on the magnetization of each domain). Each track is equipped with one or more Magnetic Tunnel Junction (MTJ) devices used as access ports for read/write operations. Conversely to conventional 2-D array-based RAM architectures, where the single bit information can be retrieved by selecting the specific cell and connecting it to the bit line, accessing the desired cell within a track requires shifting all the magnetic domains to align the information to the access port. Due to this unique characteristic, using RTM in practical environments poses new challenges, since both latency and energy per access are not deterministic, but they depend on the position of the accessed domain with respect to the access port.

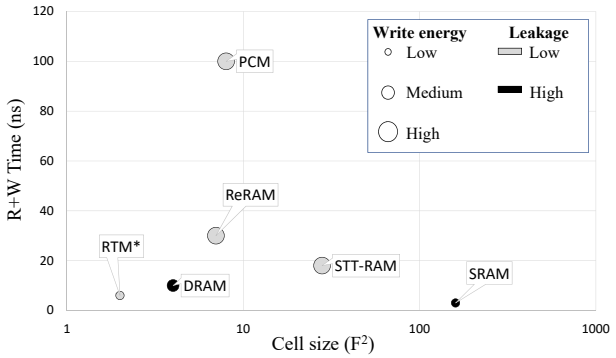

Fig. 2. Comparing different memory technologies (*shift latency not accounted).

With the aim to reduce the impact of shift operations on latency and energy consumption, RTM designers can act at different levels, either by changing the length and the shape of the tracks [8], the number [9] and the management of access ports [10], or varying the port update strategy [7] and the data placement policy [11]. However, leveraging the full potential of RTMs requires accounting for several trade-offs [12]. As an example, increasing the number of access ports allows reducing the average shift latency at the cost of a considerable area overhead, due to the use of multiple MTJs. It follows that examining performances of different memory configurations and choosing the most proper one, also depending on the target application, requires a rapid evaluation framework. Existing solutions rely on software environments, where the memory device is first modeled using tools such as *CACTI* [13] or *Destiny* [14] to extract simulation-independent parameters; then, software simulators, like *NVMain* [15] and *gem5* [16], are exploited to analyze the memory access statistics over a suite of benchmarks. However, such traditional approaches, being based on slow cycle-level simulations, do not meet the aforementioned requirements of rapid evaluation. In addition, these environments hinder the simulation of hybrid memory sub-systems [17].

To fill this gap, in this paper we present ERMES, a new emulation system for RTMs based on heterogeneous FPGA-CPU Systems-on-Chips (SoCs) platforms. The latter is an excellent candidate for emulation scopes, ensuring a high configurability degree and rapid evaluation. Moreover, the possibility to run benchmarks on the embedded processor gives designers a self-contained environment capable of easily testing different RTM architectures and data placement strategies. The main contributions of this paper are reported in the following.

- We present a new hardware architecture that emulates the behaviour of RTMs and efficiently uses the logic resources available within modern FPGA devices. The proposed architecture is fully configurable, thus it allows exploring the design space by changing design parameters, such as the number and the length of tracks and the number of access ports per track. Then, after configuration, it can be used to evaluate different data placement strategies without requiring any further hardware modification.
- The proposed emulator includes an on purpose designed memory controller that manages bit shifts in both directions optimizing the data access times.
- Result of experiments performed running several application benchmarks from the MiBench suite [18] are also presented and discussed to demonstrate the high flexibility and fast runtimes of ERMES. To the best of our knowledge, this is the first work proposing a complete FPGA-based emulator for RTMs.

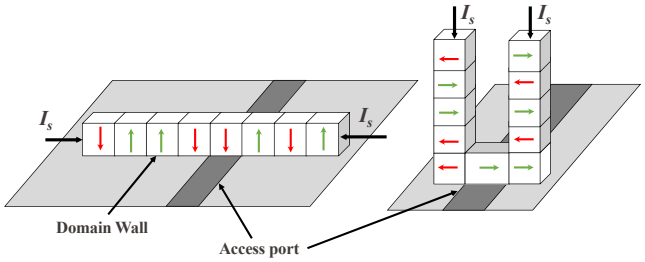

Fig. 3. Racetrack horizontal and vertical placement on a silicon wafer.

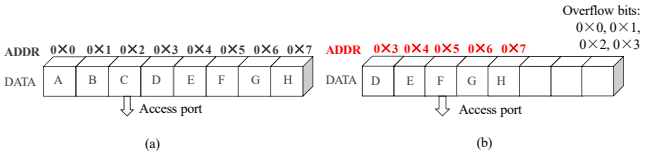

Fig. 4. Example of racetrack as shift register: (a) initial status; (b) after accessing the position 0×5.

II. Background and Motivation

A. Racetrack memory

An RTM is a three-dimensional structure consisting of magnetic nanowires that are placed horizontally or vertically on the surface of a silicon wafer, as depicted in Fig. 3. Within each nanowire (or track), multiple magnetic domains are separated by domain walls. The magnetization state of each domain (i.e. pointing up or down) serves as bit, allowing to represent 0 and 1, respectively. To perform a read operation, firstly, shift current pulses I_s are used to exert spin-transfer torque, thus shifting all magnetic domain walls along the track in the same direction. When the desired information has reached the access port, the magnetization state is determined by the MTJ device. Conversely, for a write operation, a larger current pulse is injected through the access port, thus switching the magnetization state of the corresponding magnetic domain wall. Obviously, since the access ports can perform only-read, only-write or read-write operations, the corresponding MTJ device has to be properly sized for the specific situation.

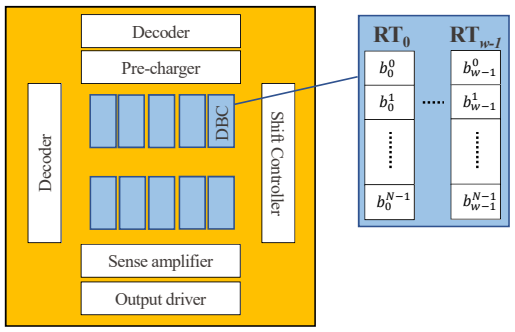

Fig. 5. RTM architecture.

From a logical point of view, tracks behave as shift registers. The number of shifts required to perform an access is determined by the distance between the address to be accessed and the position of the access port. As a consequence, at any access, the content of all magnetic domains in a track changes, with the access port serving as head whose status is dynamically updated to the last accessed domain address. Fig. 4 illustrates an example of this running for a track composed by N=8 domains and one access port originally aligned to the address 0×2 (Fig. 4(a)) . When the bit at the position 0×5 is requested, three left shifts are needed to align the bit-data F to the access port. The track content is therefore updated as reported in Fig. 4(b), while the track head is set to 0×5. It is worth noting that, in tape-shaped RTMs, some bits, usually referred to as overflow bits, may be lost because of the shifts occurring at the outermost positions. To avoid this, the number of domains in the nanowire could be increased to twice the number of bits effectively stored [10]. Even though such a solution has no significant influence on the overall area, which is mostly determined by the MTJ transistors, it affects the average read/write access time because of the increased length and number of possible shifts. An effective alternative is given by ring-shaped RTMs [8], which link the outermost positions (i.e. 0 and N-1) to provide a toroidal storage without additional domains.

To reduce the average shift latency, RTM tracks can be equipped with more than one access port. Depending on the adopted management policy, the shift controller establishes which port is in the most favourable position to access the data, computes the required number of shifts and updates the status of all access ports. Basically, two access policies are currently adopted [7]. The former, named *static*, is based on a fixed assignment of a set of domains to each port. In such a case, the logic implemented by the shift controller is quite simple because each domain may be accessed only by its assigned access port and no further elaboration is needed. The latter, referred to as *dynamic*, is thought to optimize the shift latency. In such a case, the controller firstly computes the number of shifts needed to align each access port to the desired domain and chooses the access port leading to the shortest path.

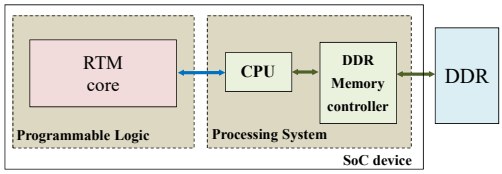

Fig. 6. Top-level architecture of ERMES.

The sequential nature of RTMs poses several challenges in how data should be organized within the memory architecture. It is typically structured as shown in Fig. 5 and contains an array of basic building blocks named Domain Block Clusters (DBCs). Each DBC accommodates several tracks, any of which can be used to store either an entire w-bit data word or single bits belonging to different words. However, the first approach leads to considerable performance degradation since multiple cycles are needed to access a single word. On the contrary, as schematized in the inset of Fig. 5, where the generic $b_i{}^j$ indicates the i-th bit of the j-th word, the second approach allows accessing all the w bits in parallel and moving all the tracks within the DBC in a lockstep fashion [7]. In this case, with M and N being the number of DBCs and the length of each track, respectively, the RTM architecture illustrated in Fig. 5 can store $M \times N$ w-bit data words.

B. Motivation of this work

Designing memory sub-systems based on RTMs requires facing new and challenging issues because of their unique characteristics. Current design aid tools based on software simulation allow a cycle-by-cycle evaluation based on specific models of the memory architecture to be performed. However, due to their long simulation times (i.e. in the order of a few hundred minutes per simulation [19]), they hinder the designers in examining different memory configurations and their impact on different benchmarks. Furthermore, such traditional aid tools do not fit well with the design of hybrid memory subsystems [20]- [21], composed by both dynamic/static RAMs and RTMs. For these reasons, hardware-based emulation platforms equipped with configurable controllers and suitable to facilitate the communication between heterogeneous memories are essential.

In the recent past, FPGA-based emulation platforms for emerging NVMs have received a great deal of attention [17], [22]- [24]. These platforms have been proposed mainly with the aim of analyzing the asymmetric access times that most NVMs suffer from. They typically use a memory controller responsible for managing memory requests, one or more delay modules, which emulate delay injection, and an external DRAM that acts in place of the NVM. The above cited emulators may be exploited for the evaluation of STT-RAMs, ReRAMs, PCMs and other NVM architectures based on conventional 2D arrays. However, they cannot be useful in the case of RTMs, since there the access to a specific

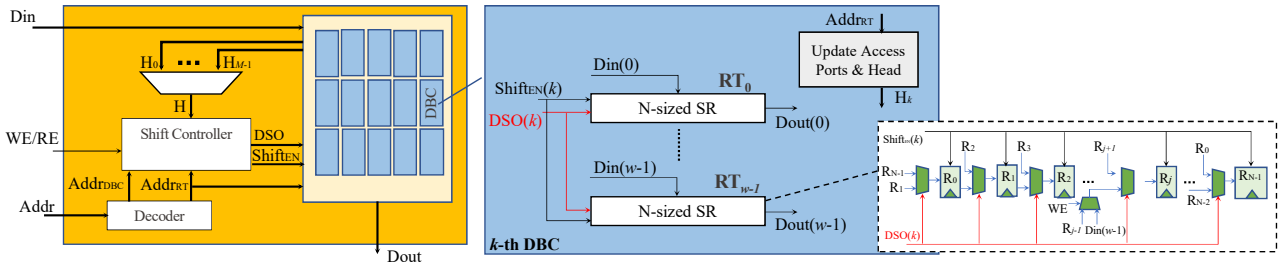

Fig. 7. Internal architecture of the *RTM core*.

```
1:  INPUT: Address ART
2:  OUTPUT: shift direction DSO, number of shifts NSO
3:  PreviousJump ← 0;
4:  for i from 0 to P-1 do
5:      H(i)← INITHEAD(i); Jump(i) ← ART – H(i);
6:      if |Jump(i)|≤ N/2 do
7:          NS(i) ←|Jump(i)|;
8:          if Jump(i) <0 do //Right direction
9:              DS(i) ←1; else //Left direction
10:             DS(i) ←0;
11:     else
12:         NS(i) ←N-|Jump(i)|;
13:         if Jump(i) <0 do // Left direction
14:             DS(i) ←0; else // Right direction
15:             DS(i) ←1;
16: [NSO, Id] ←min(NS(1:P)) ; // NSO=min value, Id=min position
17: DSO ← DS(Id); PreviousJump ← (-1)^DSO ×NSO;
18: for i from 0 to P-1 do
19:     if i = Id do
20:         H(i)← ART; else
21:         H(i)← H(i) + PreviousJump;
```

Fig. 8. Pseudo code describing operations performed by the shift controller.

position results in a change in all the other domains within the track, and this influences the following accesses. For this reason, RTM emulators must keep trace of how domains have been moved after any read/write access, besides managing its variable latency. Based on these considerations, this work presents for the first time an efficient and flexible FPGA-based emulation platform for RTMs.

III. The Proposed ERMES Design

Fig. 6 shows the top-level architecture of the proposed emulation system. It includes the Processing System (PS) and the Programmable Logic (PL) sections. The CPU in the PS runs the benchmark software routines and generates memory requests to be forwarded to the configurable *RTM core* implemented in the PL. The *RTM core* communicates with the PS through the AXI-Lite protocol. It outputs the number of shifts needed to access the requested address, a validation signal to indicate that the access has been correctly executed, and eventually the read data. The PS also communicates with an external Double Data Rate (DDR) DRAM memory to support the evaluation of hybrid memory sub-systems.

A. RTM core

The custom *RTM core* has been designed by the Very High-Speed Integrated Circuits Hardware Description Language (VHDL) at the Register-Transfer-Level of abstraction, taking into account the memory architecture described in Section II. Depending on the user's requirements, the number of DBCs M, the number of tracks per DBC (i.e. the data word size w) and the number of positions per track N can be easily configured. Moreover, the number of access ports per track P is chosen by the designer.

Its internal architecture is illustrated in Fig. 7; it consists of a decoder, a shift controller and M DBCs. The access request sent by the CPU includes the address to be accessed (i.e. $Addr$), the type of operation to be performed (i.e. write WE or read RE), and the w-bit Din word to be written. The $Addr$ signal is represented on y bits, with $y = \lceil log_2(M \times N) \rceil$; its most significant bits ($Addr_{DBC}$) identify the selected DBC, whereas the least significant bits ($Addr_{RT}$) identify the specific position in the racetracks. Then, the shift controller establishes the minimum latency direction (DSO) to fulfil the access request and the expected number of shifts. To this aim, the shift controller has to also consider the current alignment of the access ports with the racetrack cells. An M:1 multiplexer is therefore employed to select the H set referred to the addressed DBC. The operations performed by the controllers to manage write and read accesses, as well as port updating, are reported in the pseudo-code of Fig. 8. Here $INITHEAD$ represents the initial alignment of the P ports within the N-sized track. The shift controller computes the number of shifts required by each port to access $Addr_{RT}$ (lines 5-15). Then, it computes the number of shifts NSO and the direction DSO of the port leading to the shortest path (i.e. port Id). Such an information is transferred to the CPU to evaluate the access latency. It is worth noting that, after the alignment between the access port and the requested bit-data has been reached, the status of all the ports must be updated. Therefore, while the current head of the selected port $H(Id)$ is updated with the accessed address, heads of the other ports are computed taking into account how many shifts have been performed in the previous access request (line 21).

To emulate the behaviour of a bidirectional shift register, each track within the DBC has been described as N registers connected to as many multiplexers. Depending on the selector DSO, the generic register R_j receives the bit-data on either its left (R_{j-1}) or its right (R_{j+1}). The $Shift_{EN}$ signal coming from the shift controller is then driven high for a time

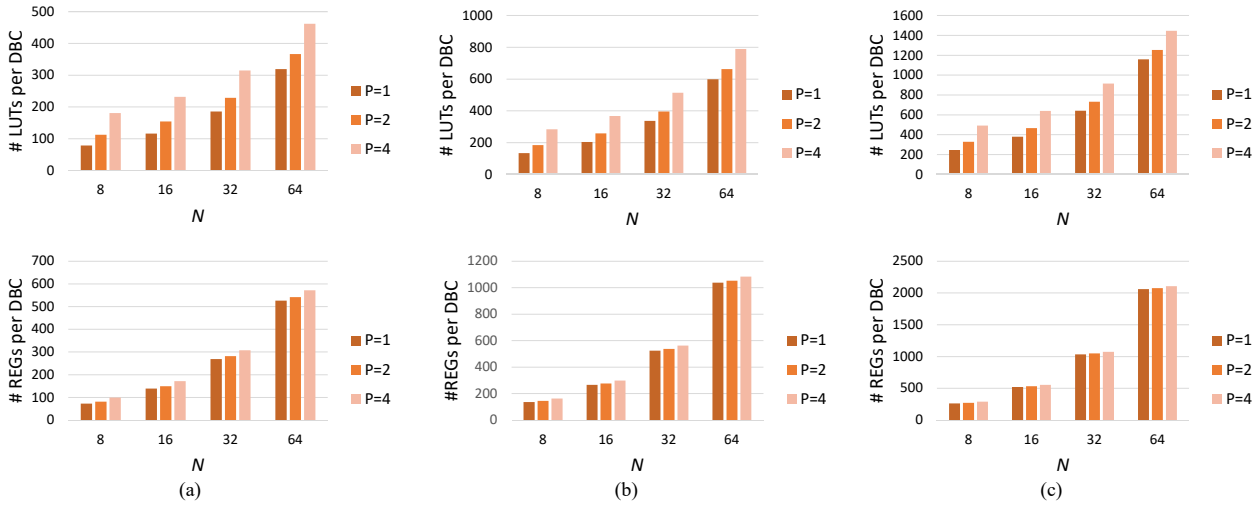

Fig. 9. Analysis of ERMES hardware requirements for different DBC configurations: (a) w=8; (b) w=16; (c) w=32.

long enough to perform the computed number of shifts, thus allowing the requested bit-data to be aligned with the selected access port.

B. Interfacing CPU and RTM core

Application benchmarks used to test the memory sub-system are executed by the CPU embedded within the PS. Depending on the requested memory address, the CPU establishes if it has to be transferred the to either the external memory or to the *RTM core* implemented in the PL. In the latter case, an address re-mapping is done to ensure that the application memory request complies with the address space assigned to the custom *RTM core* connected through the AXI-Lite protocol. To extend interoperability with other system-simulation environments, ERMES can be easily interfaced with an external host general-purpose processor. This option just requires enabling the PCI Express (PCIe) interface implemented by either hard or soft IP blocks, depending on the target FPGA platform.

IV. Results and Discussion

For evaluation purposes, ERMES has been implemented on the Xilinx ZCU102 development board equipped with the Zynq XCZU9EG heterogeneous SoC. The overall system has been characterized using the Xilinx Vivado 2021.2 development tool, imposing a timing constraint of 10 ns for the PL running frequency. In this section, we first present hardware implementation results achieved by different DBC configurations. Then, we analyze the impact of some design settings, like M, N, and P, and different data placement policies on the latency performance exhibited by the proposed emulation system when running sample benchmarks from the MiBench suite [18].

A. FPGA implementation

As described in the previous section, each DBC within the *RTM core* architecture contains w N-sized tracks and one module responsible for updating the status of the P access ports. Depending on such design parameters, different configurations may be realized. Fig. 9 shows the impact of varying N, w and P over the number of occupied Look-Up-Tables (LUTs) and registers per DBC. At a glance, it can be observed that, at a parity of P, the number of LUTs and registers increases with a different rate at the growing of the DBC storage capacity $w \times N$. As expected, the number of registers is proportional to such capacity, with slight deviations due to the contribution of the module implementing the access ports updating. On the contrary, logic resources needed to implement shift registers are proportional to $N/2$. This is due to the synthesis optimizations made on the generic N-sized track to pack more multiplexers within the same LUT. As visible in Fig. 9, the number of occupied LUTs is significantly dependent on P. Eqs. (1)-(3) provide analytical estimation of the amount of required LUT and FF resources as function of N, w and P. It can be seen that, while the contribution of the shift controller (i.e. LUT_C and REG_C) is mainly due to the number of access ports P and the track length N, the area occupied by a DBC (i.e. LUT_D and REG_D) is mostly influenced by its number of racetracks w.

$$\begin{aligned} LUT &= LUT_C + LUT_D, \\ REG &= REG_C + REG_D \end{aligned} \tag{1}$$

$$\begin{aligned} LUT_C &= P \times (log_2(N) + 2), \\ REG_C &= (3P + 1) \times (log_2(N) + 1) \end{aligned} \tag{2}$$

$$\begin{aligned} LUT_D &= (w \times N/2) \times P(log_2(N) + 1), \\ REG_D &= (w \times N) \times P(log_2(N) + 1) \end{aligned} \tag{3}$$

As a final remark, it is worth noting that several different configurations could lead to a specific DBC storage capacity. As an example, a DBC can store 512 bits by setting (w, N) to either (8, 64), (16, 32) or (32, 16). This significantly influences

TABLE I
APPLICATIONS SPECIFICATIONS

Application	Description	Memory footprint	Number of memory accesses	Effective RTM accesses
qsort	Quick sort algorithm on array of strings	2.47MB	648233	3762
patriciaSmall	Routing algorithm for network applications	2.53MB	663772	4194
bitcountSmall	Count the number of bits in array of integers	17.47MB	4581094	817803
stringsearchSmall	Search words into phrases	2.52MB	661964	10110
dijkstra	Analysis of graphs to solve the shortest path problem	2.47MB	649897	15639

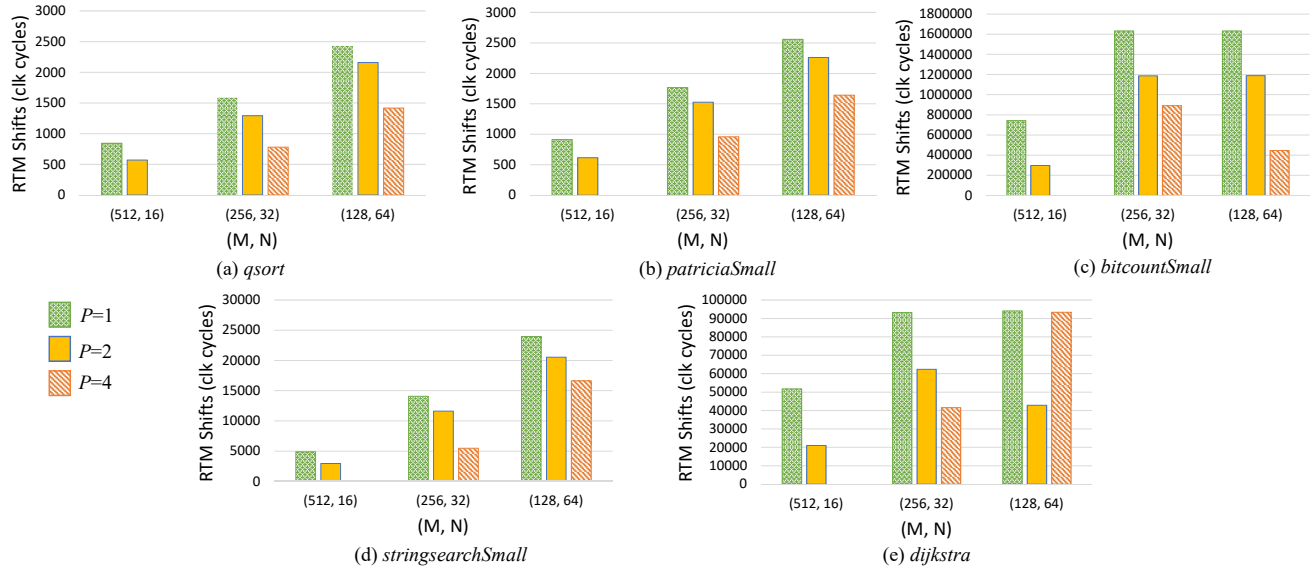

Fig. 10. Analysis of ERMES shift latency for different memory configurations and benchmarks.

the performance-area trade-off of the RTM. High w and low N values allow more bits can be accessed in parallel at a reduced average access latency. However, such an approach has the drawback of larger area occupancy [11]. Conversely, low w and high N values ensure a more compact RTM design, but they degrade the average speed performance. In such a case, other countermeasures (i.e. increasing P) can be investigated to trade-off latency and area.

B. Benchmarks evaluation

Table I summarizes the main characteristics of the evaluated applications from the MiBench suite [18]. It is worth noting that, considering the amount of hardware resources available on the XCZU9EG chip, the largest memory size that can be emulated is 32kBytes. Therefore, the CPU running the benchmarks selects a portion of memory requests, corresponding to a specific addresses space, and transfers them to the *RTM core* through the AXI-Lite interface. In such a case, the *RTM core* acts as a scratchpad backed up to the external DRAM, which is used for the remaining access requests as typically happens in hybrid NVM-DRAM memory sub-systems. The effective number of memory accesses requested to the *RTM core* is reported in Table I. Several configurations of the 32kBytes *RTM core* have been analyzed by varying the number of access ports per track P and the memory array partitioning (i.e. length of tracks N and number of DBCs M), while keeping the data word size w=32. Fig. 10 illustrates the overall number of shift operations required to run each application for the chosen configurations. As expected, increasing the length of the tracks at a parity of P generally leads to higher number of shift operations. As an example, when P=1, moving N from 16 to 32 and 64 impacts on the total number of shift operations by $\times 2.13$ and $\times 2.91$ factors, on average, respectively. Conversely, at a parity of N=32, increasing the number of access ports P from one to two and four contributes to reduce the latency, respectively, by 1.3 and 2.1 times, on average. It is worth observing the particular behavior exhibited by the *dijkstra* application, which does not appear to benefit from the increasing of the number of access ports for larger tracks (i.e. N=64). Further investigations were conducted on this benchmark and we noted that the above effect is influenced by the initial position of the access ports. In such a case, we realized that alignments with addresses lower than 0×18 lead to a 54% reduction on the shift latency. The optimization of the access port alignment is beyond the scope of this work, but it can be easily performed by using ERMES.

Table II reports hardware requirements for the 32kBytes RTM configurations analyzed, including resources employed by DBCs and shift controller. In such a case, being M=32kBytes/($4 \times N$) the number of DBCs, the total oc-

TABLE II
HARDWARE REQUIREMENTS FOR DIFFERENT 32KBYTES RTM CONFIGURATIONS

(M, N)	(512, 16)	(256, 32)	(128, 64)
P=1	180.2k LUTs, 258.5k Regs	154.5k LUTs, 257.5k Regs	141.4k LUTs, 256.8k Regs
P=2	216k LUTs, 261k Regs	172.8k LUTs, 259k Regs	150.8k LUTs, 257.7k Regs
P=4	Not implementable	209.3k LUTs, 262k Regs	169.4k LUTs, 259.5k Regs

cupancy of LUTs and registers decreases with N, while it increases when more access ports are used.

Notably, the proposed emulation system can be adopted to evaluate different data placement policies and examine the latency required by different applications for a given memory configuration. As an example, in our experiments we analysed the shift behaviour of the *RTM core* configured with M=128, w=32, N=64 and P=1 when two state-of-the-art data placement policies are used. The former, named *First Come First Store* (*FCFS*), stores data into racetracks according with the order it is required. The latter, known as *Most Access First* (*MEF*), orders data within the racetracks depending on its occurrence (i.e. from the most to the least frequently accessed data). Fig. 11 compares the average number of shifts per application benchmarks obtained by *FCFS*, *MEF* and *No strategy* (i.e. data is put within the RTM based on original trace addresses). In general,it is evident that using particular strategies on the basis of simple assumption about the data access patterns significantly alleviates the impact of shifts in RTMs. According to the literature, for the selected benchmarks, the *FCFS* strategy allows the lowest latency, which is the result of quite sequential and no-repetitive patterns. It is worth noting that such kind of investigation does not require any re-synthesis of the emulator architecture.

C. Performance Analysis

A direct comparison with state-of-the-art FPGA-based emulators [17], [22]– [24] is not possible, due to the the different supported features. As an example, the system presented in [17] is based on a host CPU that communicates with the FPGA through PCIe interface; the programmable logic just accommodates the Hybrid Memory Manager Unit (HMMU) core, which is used to forward the memory access requests produced by the host CPU to either DRAM or NVM. In such a case, the generic NVM device is supposed to be external, which prevents from keeping trace of how information are moved after any read/write access in the case of RTMs. Conversely, the FPGA-based platforms [22], [23] and [24] emulate the NVM device through the DRAM. To this aim, they model the asymmetric read/write latency of NVMs through controlled delay-injection modules. However, being based on DRAM, this emulation environment is suitable only for NVM technologies based on conventional 2D arrays (i.e. STT-RAMs, ReRAMs and PCMs). Therefore, we evaluated ERMES speed-up over software-based simulators [25] running on the Intel

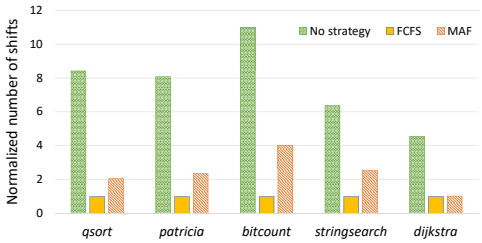

Fig. 11. Analysis of shift latency for different data placement policies.

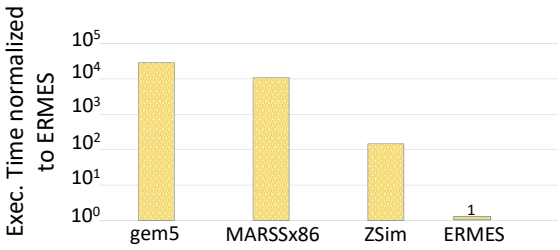

Fig. 12. Normalized simulation/emulation time running applications from the MiBench suite.

i7-477 core. Fig. 12 reports a comparison in terms of average simulation times for the MiBench applications. In comparison to the cycle-level *gem5* and *MARSS*×86 simulators, which imitate the operations of the simulated memory for each cycle, the proposed emulation system experiences an ×28000 and ×10000 speedup, respectively. Parallel simulators like *ZSim*, being based on multi-thread processes, significantly reduce simulation times with respect to the cycle-level counterparts. Nonetheless, thanks to its on-purpose designed architecture, ERMES exhibits an ×146 speed-up over *ZSim*.

Finally, ERMES latency results are used to analyze the actual speed-up achieved by RTMs over other traditional NVM technologies, for the applications mentioned above. It is worth noting that, thanks to their constant access latencies, SRAM and DRAM devices achieve average performances at least 2.6 times higher than the RTM counterpart for the same set of benchmarks. However, as above mentioned, such CMOS technologies suffer for high leakage and considerable area overhead, which make them unattractive for low-energy and area-constrained applications. Fig. 13 shows the total access time exhibited by the ReRAM, STT-RAM and RTM technologies normalized to the slowest PCM counterpart, taking into account data reported in [21]. The chosen architecture for the RTM is M=512, N=16 and P=1. It is evident that the RTM outperforms competitors in all evaluated benchmarks; this is because the additional shift latency accounts for only 10% of the total access time, on average.

V. CONCLUSIONS AND FUTURE TRENDS

Thanks to their low-area and low-energy performances, emerging RTMs are effective candidates for current IoT edge devices. However, since such memories involve shifting of magnetic domains to retrieve information within the tracks

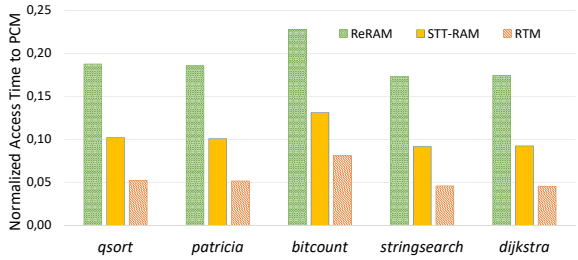

Fig. 13. Normalized access time for the analyzed benchmarks on different NVM technologies.

and align it to the access port, the latency per operation is not deterministic. This unique characteristic revolutionizes the design of such memory sub-systems, since several application-related optimizations can be introduced either on architecture or data placement or both to reduce the impact of shift operations. Current design aid tools based on software simulations offer a cycle-by-cycle evaluation on specific memory models, with prohibitive simulation times and limited flexibility. In this paper we presented ERMES: an emulation system based on heterogeneous FPGA-CPU Systems-on-Chips for fast and accurate analysis of RTM architectures. The proposed configurable platform assists in designing RTM architectures and evaluating the impact of design choices and data placement policies over the latency. Experiments conducted on benchmark applications from the MiBench suite show that our RTM emulator implemented within the Zynq XCZU9EG SoC is significantly faster than the *gem5* software simulator. Finally, the results obtained in terms of emulation speed and prototyping flexibility stimulate investigation of new approaches for data placement and access policies, also based on approximate operations. In order to encourage research in this direction and provide an effective support, ERMES will be made open-source at *https://cfaed.tu-dresden.de/pd-downloads*.

REFERENCES

[1] J.P. Kulkarni, J.W. Tschanz, V.K. De, "Energy Efficient Volatile Memory Circuits for the IoT Era," in Enabling the Internet of Things, M. Alioto, Eds. Springer, 2017, pp. 149-170.

[2] A.I. Alsalibi, M.K.Y. Shambour, M.A. Abu-Hashem, M. Shehab, Q. Shambour, R. Muqat, "Nonvolatile Memory-Based Internet of Things: A Survey," in Artificial Intelligence-based Internet of Things Systems, S. Pal, D. De, and R. Buyya, Eds. Springer, 2022, pp. 285–304.

[3] M. Hosomi, H. Yamagishi, T. Yamamoto, K. Bessho, Y. Higo, K. Yamane, H. Yamada, M. Shoji, H. Hachino, C. Fukumoto, H. Nagao, H. Kano, "A novel nonvolatile memory with spin torque transfer magnetization switching: Spin-RAM," in Proc. IEEE Int. Electron Devices Meet. (IEDM), 2005, pp. 459–462.

[4] H.-S.P. Wong, S. Raoux, S. Kim, J. Liang, J.P. Reifenberg, B. Rajendran, M. Asheghi, K.E. Goodson , "Phase change memory," *Proc. IEEE*, vol. 98, no. 12, pp. 2201–2227, Dec. 2010.

[5] Y. Chen, "ReRAM: History, Status, and Future," *IEEE Trans. Electron Devices*, vol. 67, no. 4, April 2020, pp. 1420–1433.

[6] S. Parkin and S.-H. Yang, "Memory on the racetrack," *Nature Nanotechnology*, vol. 10, pp. 195–198, Mar. 2015.

[7] R. Bläsing, A.A. Khan, P. CH. Filippou, C. Garg, F. Hameed, J. Castrillon, "Magnetic Racetrack Memory: from Physics to the Cusp of Applications within a Decade," *Proc. IEEE*, vol. 108, no. 8, pp. 1303–1321, Aug. 2020.

[8] G. Wang, Y. Zhang, B. Zhang, B. Wu, J. Nan, X. Xhang, Z. Zhang, J.-O. Klein, D. Ravelosona, Z. Wang, Y. Zhang, W. Zhao, "Ultra-Dense Ring-Shaped Racetrack Memory Cache Design," *IEEE Trans. Circ. Syst. I: Reg. Pap.*, vol. 66, no. 1, pp. 215–225, Jan. 2019.

[9] A.A. Khan, F. Hameed, R. Bläsing, S. Parkin, J. Castrillon, "RTSim: A Cycle-Accurate Simulator for Racetrack Memories," *IEEE Comput. Architec. Lett.*, vol. 18, no. 1, pp. 43–46, 2019.

[10] R. Venkatesan, V. Kozhikkottu, C. Augustine, A. Raychowdhury, K. Roy, A. Raghunathan, "TapeCache: A High Density, Energy Efficient Cache Based on Domain Wall Memory," in Proc. 2012 ACM/IEEE Int. Symp. Low Power Elect. Design(ISLPED '12), July 2012.

[11] A.A. Khan, A. Goens, F. Hameed, J. Castrillon, "Generalized data placement strategies for racetrack memories," in Proc. 2020 Design, Automation and Test in Europe Conference and Exhibition (DATE), Mar. 2020.

[12] S. Mittal, "A Survey of Techniques for Architecting Processor Components using Domain-Wall Memory," *ACM J. Emerg. Techn. Comput. Syst.*, vol. 13, no. 2, article 29, Nov. 2016.

[13] S.J.E. Wilton and N.P. Joupp, "CACTI: an enhanced cache access and cycle time model," *IEEE J. Solid-State Circuits*, vol. 31, issue 5, pp. 677–688, May 1996.

[14] M. Poremba, S. Mittal, D. Li, J.S. Vetter, Y. Xie, "DESTINY: A Tool for Modeling Emerging 3D NVM and eDRAM caches," in Proc. 2015 Design, Automation and Test in Europe Conference and Exhibition (DATE), Mar. 2015.

[15] M. Poremba and Y. Xie, "NVMain: An Architectural-Level Main Memory Simulator for Emerging Non-volatile Memories," in Proc. 2012 IEEE Computer Society Annual Symposium on VLSI, Aug. 2012.

[16] 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, D.A. Wood, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, issue 2, pp. 1-7, May 2011.

[17] F. Wen, M. Qin, P. Gratz, N. Reddy, "An FPGA-based Hybrid Memory Emulation System," in Proc. 2021 31st International Conference on Field-Programmable Logic and Applications (FPL), Aug. 2021, pp. 190–196.

[18] M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, R.B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in Proc. Fourth Annual IEEE International Workshop on Workload Characterization (WWC-4 Cat. No.01EX538), Dec. 2001.

[19] K. Iordanou, O. Palomar, J. Mawer, C. Gorgovan, A. Nisbet, M. Lujàn, "SimAcc: A Configurable Cycle-Accurate Simulator for customized accelerators on CPU-FPGAs SoCs," in Proc. IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), Apr. 2019, pp. 163-171.

[20] A.A. Khan, N.A. Rink, F. Hameed, J. Castrillon, "Optimizing Tensor Contractions for Embedded Devices with Racetrack and DRAM Memories," *ACM Trans. Embed. Comput. Syst.*, vol. 19, no. 6, article 44, Sept. 2020.

[21] A.L. Takami, A. Hoseinghorban, M. Bazzaz, A. Ejlali, "RIDE: Energy Efficient Data Allocation on Compound Racetrack-SRAM Scratchpad Memory for Real-Time Embedded Systems," in Proc. 2020 CSI/CPSSI International Symposium on Real-Time and Embedded Systems and Technologies (RTEST), June 2020.

[22] A.K. Jain, S. Lloyd, M. Gokhale, "Performance Assessment of Emerging Memories Through FPGA Emulation," *IEEE Micro*, vol. 39, issue 1, pp. 8-16, Jan.-Feb. 2019.

[23] T. Lee and S. Yoo, "An FPGA-based platform for non volatile memory emulation," in Proc. 2017 IEEE 6th Non-Volatile Memory Systems and Applications Symposium (NVMSA), Aug. 2017.

[24] Y. Omori and K. Kimura, "Non-Volatile Main Memory Emulator for Embedded Systems Employing Three NVMM Behaviour Models", *IEICE Trans. Information and Systems*, vol. E104.D, no. 5, pp. 697–708, May 2021.

[25] A. Akram and L. Sawalha, "A Survey of Computer Architecture Simulation Techniques and Tools," *IEEE Access*, vol. 7, pp. 78120-78145, 2019.