Architecting Racetrack Memory preshift through pattern-based prediction mechanisms

***Abstract*— Racetrack Memories (RM) are a promising spintronic technology able to provide multi-bit storage in a single cell (tape-like) through a ferromagnetic nanowire with multiple domains. This technology offers superior density, non-volatility and low static power compared to CMOS memories. These features have attracted great interest in the adoption of RM as a replacement of RAM technology, from Main memory (DRAM) to maybe on-chip cache hierarchy (SRAM). One of the main drawbacks of this technology is the serialized access to the bits stored in each domain, resulting in unpredictable access time. An appropriate header management policy can potentially reduce the number of shift operations required to access the correct position. Simple policies such as leaving read/write head on the last domain accessed (or on the next) provide enough improvement in the presence of a certain level of locality on data access. However, in those cases with much lower locality, a more accurate behavior from the header management policy would be desirable. In this paper, we explore the utilization of hardware prefetching policies to implement the header management policy. "Predicting" the length and direction of the next displacement, it is possible to reduce shift operations, improving memory access time. The results of our experiments show that, with an appropriate header, our proposal reduces average shift latency by up to 50% in L2 and LLC, improving average memory access time by up to 10%.**

Keywords—Racetrack Memory, Cache Hierarchy, Header Management

I. INTRODUCTION

Cache Memories occupy a growing fraction of transistor count (and chip area) in modern processors. It seems that the demand for even larger on-chip storage will continue in the near future, driven by the increasing processor-memory performance gap and the growing datasets of emerging applications [1]. Consequently, there is a great interest in emerging memory technologies able to provide higher density and better energy efficiency. Racetrack memories [2][3][4][5] are promising spintronic-based non-volatile memories, which combine the speed of SRAM, the density of DRAM and the non-volatility of Flash memory. This technology can provide even larger integration density than alternative emerging technologies, such as spin-transfer torque random-access memory (STT-RAM) [6] or phase-change memory (PCM) [7].

A Racetrack Memory cell consists of a ferromagnetic wire where electron spins are employed to encode binary data information. Its early implementations, known as Domain Wall Memories or DWM [2] encoded information by a train of spin-up or spin-down magnetic domains separated by Domain Walls. More recently, it has been demonstrated that nanometer-scale skyrmions [8] can also be employed to encode information in a metalic racetrack, providing higher package density, lower energy and more robust data stability [5]. As seen in Fig. 1, each RM cell is able to store multiple data bits in a single wire programming domains to a certain direction (DWM) or by the absence or presence of a skyrmion (SK-RM). Applying a current through the wire ends, domains or skyrmions can be shifted left/right at a constant velocity. With such a tape-like operation, every domain can be aligned with a read/write port, implemented through a Magnetic Tunnel Junction (MTJ). The bit-cell structure required for shifting and read/write is shown in Fig. 1.down. Read/write operations are performed precharging bitlines (BL and BLB) to the appropriate values and turning on the access transistors (T_{RW1} and T_{RW2}). Bit shifting requires an additional pair of transistors connected to the edges of the nanowire. During a shift, transistors T_{S1} and T_{S2} are turned ON, while BL and BLB lines are connected to Vdd/ground (depending on the shift direction. Left shift: BL=Vdd, BLB=ground).

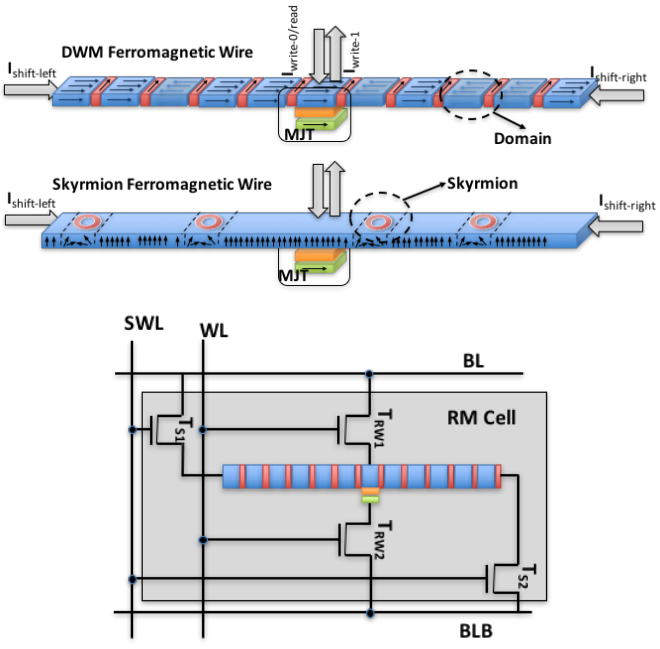

Fig. 1. (up) Racetrack Memory structure, Domain Wall Memory and Skyrmion Memory. (down) Bit-cell structure.

From the architecture side, the main drawback of this kind of memory technology is the variable access time. Accessing a bit stored in the metalic wire involves two steps: aligning the domain/skyrmion to an access port and performing the read or write access. As alignment depends on the current position of

the domain/skyrmion to be accessed relative to the access port, access latency is variable. Such a property is undesired, especially in higher levels in the memory hierarchy. It can be mitigated through the inclusion of additional read/write ports [9][10], being the main drawback of these solutions the area overhead [11][12], which could diminish the density benefits. An alternative solution consists on the utilization of prediction mechanisms able to proactively perform domain alignment [11], moving domains/skyrmions prior to memory access. Previous works have already demonstrated the benefits of simple header management policies in the presence of data locality [13], simply leaving access port in a position close to the last access. However, in the absence of locality these mechanisms might not be accurate enough to minimize the overhead of header alignments. In this work, we explore the suitability of techniques based on pattern recognition for header management, focusing on Racetrack Memory implementations with single read/write port. We propose and evaluate a new preshifting policy relying on correlation-based prefetching. We evaluate multi-domain racetrack memories as part of the levels of memory hierarchy above LLC (which correspond to private cache levels), comparing its performance to state-of-the-art counterparts and evaluating the proposal through full-system simulation. We make use of a large number of workloads, belonging to multiple benchmark suites.

The rest of the paper is organized as follows: Section II describes related work on RMs, focusing on header management policies, Section III presents the proposed cache architecture, its organization and a working example, Section IV explains the evaluation methodology, Section V explores the optimal configuration for the proposed mechanism and Section VI is devoted to the performance evaluation. Finally, Section VII states our main conclusions.

II. Related Work

The potential benefits of this kind of technology have made Racetrack Memories an active research field in the recent years. Multiple works have demonstrated alternative prototypes of this technology [14][15][4][16] and many efforts have been recently involved in skyrmion-based RMs research and development [17][5][8][18], given their unique properties [19]. These promising results have inspired many authors to explore the utilization of RMs as a candidate to replace CMOS-based DRAM/SRAM.

The author in [20] recently presented an extensive survey of architectural techniques for using RM. For this reason, this section will only focus on alternative header management policies found in the literature, more similar to our proposal. The architectural proposals aimed at mitigating the latency overhead caused by header alignment follow two different approaches: moving data closer to the read/write header [21] or implementing read/write header alignment policies able to minimize access latency [22][11][13][23][24][25].

A simple and efficient data movement policy relies on the migration of frequently accessed blocks closer to the access port [22][11]. Additionally, some RM-oriented cache implementations also make use of data migration as part of their behavior. In [12], a merged two-level cache is proposed, in which migration is required to move the LRU block to the domain closest to the read/write port. Cache banks mixing single-domain and multi-domain cells [11] make use of similar migration mechanisms, moving LRU blocks to single-domain cells.

To the best of our knowledge, the only preshifting proposal for general purpose architectures consists of next-block preshifting [13][11]. Our proposal moves one step further and evaluates more sophisticated mechanisms, making use of past access patterns (shift size and direction) to predict the next shift operation.

III. Preshift RM cache architecture

This section provides an overview of the architecture proposed, describing key features such as data organization, addressing policy and header management. We focus our attention on a 3-level on-chip cache hierarchy, which is commonly used by current processors. The configuration used has two private levels and a last level shared among all system cores. Data and Instructions are divided at the first level of private caching, sharing the same storage in the rest of levels. We assume the utilization of RM technology in all levels. 1-domain [1] cells [26] are employed in those latency-critical elements (L1 Instruction cache, Tag Array of those banks with sequential access). The use of multi-domain cells is limited to L1D (data and tag arrays) and Data Arrays of L2/LLC cache blocks.

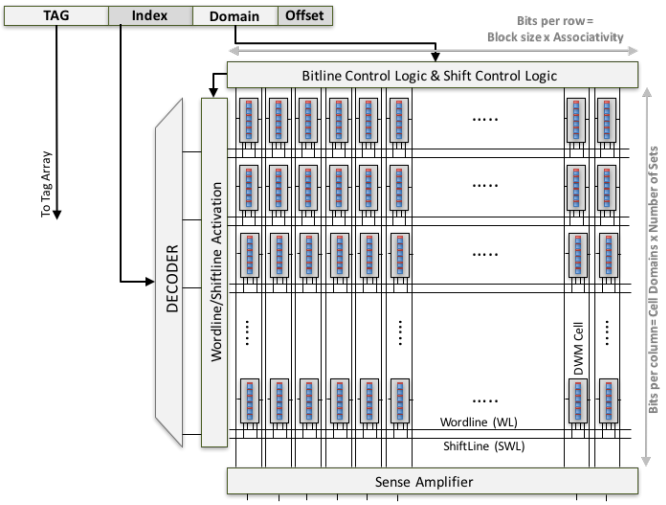

Fig. 2. Address bits (above) and cache organization (below)

A. Bank/Data Organization

Fig. 2 sketches the data array organization of the architecture employed for Multi-domain cache banks, similar to the one proposed in [13]. RMs are arranged "vertically", assigning each domain to a different set of the bank. All the RMs in the same row share the same header alignment and perform shift, read and write operations simultaneously. To store a cache block 512 cells (64B per block) are used and each domain inside the

[1] For SK-RM, we define "Domain" as the fraction of wire required to store a single skyrmion. This way, the number of domains describes the wire capacity (number of skyrmions that can be created in the wire).

nanowire contains a bit from a different block (e.g., domain 0 stores a bit from set 0, domain 1 stores a bit from set 1 and so on). This implementation makes use of the RM cell described in Fig. 1.c as the basic building block. The row decoder drives both *Wordlines* (WL) and *Shiftlines* (SL), employing the logic in *Wordline*/*Shiftline* Activation to select the kind of operation to perform (read/write or shift). Column logic controls *Bitlines* (BL and BLB) to carry out read, write and shift operations (only modifying Voltage values). Tag and offset bits are used in the same way as a conventional SRAM cache bank. In contrast, index bits are divided into Cell bits and Domain bits. Cell bits (labeled as Index) are used to select a RM-cell row and domain bits are used to select a singular domain inside the RM-cell. Therefore, each row of RM-cell will store a group of consecutive sets in the cache. Shift Control Logic is in charge of storing the header alignment of each RM row and calculating the appropriate shift operation according to the incoming index bits.

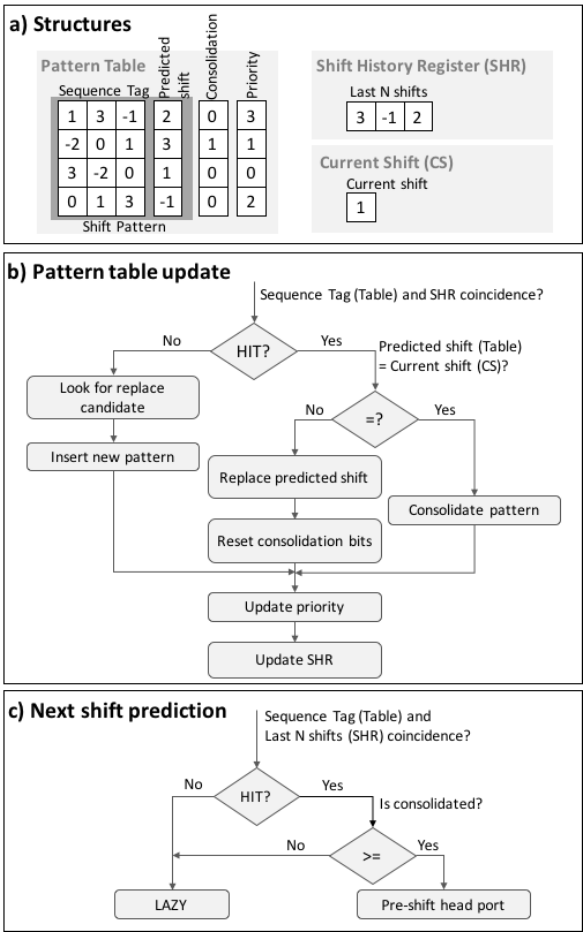

Fig. 3. (a) Hardware structures required for pattern-based prediction. (b) Pattern table update after each cache access. (c) Next shift prediction according to Pattern table values.

B. Header Management

The accuracy of the header management policy for state-of-the-art proposals is highly dependent on the presence of temporal and spatial locality. In this context, Lazy policy [13], which leaves the head port in the last accessed domain, exhibits good performance. Unfortunately, the locality characteristics vary with application and/or hierarchy level. In order to compensate for this heterogeneity, we propose a hybrid head management policy, combining a policy with a preshifting mechanism based on cache access pattern recognition with the Lazy approach.

The information employed for pattern identification and prediction is *the distance* between consecutive accesses. Shift distance is always obtained as the difference in steps between the domains referred to by two consecutive accesses. Shift information includes both dimension (distance between domains) and direction (left or right shift). It is also feasible to define each element of access pattern (such as domain number, or memory address). For now, we focus on shift patterns for space exploration, leaving alternative mechanisms for Sub-Section 5.1.

The proposed mechanism makes use of two hardware structures, a Pattern Table and a Shift History Register (SHR). Additionally, the current shift associated to the current access is available through Current Shift (CS). The SHR register keeps the W most recent shift operations in the whole bank. There is one entry in the table for a set of all the possible W most recent shift patterns. The internal structure of the Pattern Table is shown in Fig. 3, and contains the following fields that tracks the recent past access pattern of that group of sets:

- Shift Pattern: information about W past consecutive shifts accesses (W=3 in Fig 3.a). Each entry is made up of the Sequence tag (1,3,-1 first entry) and the Predicted Shift (2). The SHR value (last W bits) is compared to each Sequence tag values to perform prediction. Predicted shift is the value used for preshifting. Sequence Tag length depends on pattern length whereas Predicted shift only stores one value.
- Consolidation: Number of times a pattern repeats.
- Priority: if table size is not enough to store all possible patterns, policy replacement uses this field to choose the pattern to evict. This policy mimics the LRU algorithm employed for way replacement in associative caches.

On every cache access the mechanism will sequentially perform the following two operations:

Pattern Table Update. Fig. 3.b sketches the steps involved in the process of updating the *Pattern Table* after each access. The current value of *SHR* is looked up in the *Pattern Table*. On a hit, *Current Shift* and *Predicted Shift* field values are compared. If the two match, the *Consolidation* value is incremented. If the two values differ, the *Predicted Shift* value is removed and replaced by the *Current Shift* value. In this case, *Consolidation* is also reset. If the pattern is not in the table, it must be inserted, it being necessary to evict an LRU entry. Then, both the *Priority* values and the *SHR* are updated. *SHR* is shifted to insert the *Current Shift* as part of the last N shifts.

Next Shift Prediction. Subsequently, as described inFig. 3.c, prediction will be performed. A lookup in the *Pattern Table* with the updated *SHR* will be performed. In the case of a Hit and if the *Consolidation* threshold is exceeded, the RM header is shifted to the predicted value. In any other case, the *Lazy* policy is applied, i.e. that header remains in its current position.

C. Working Example

To better understand the mechanism, next we will illustrate it with a greatly simplified configuration (Fig. 4). A nanowire with 8 domains and the head port initially located in domain 0x4 is used. A two-shift pattern length and single hit for

consolidation are employed. Pattern table contains only 4 entries. The initial status of the Pattern Table, Shift History Register and Current Shift is shown in the first box in Figure 4. For the initial state of the Pattern Table, we describe the evolution of header position, shift prediction and table content for the following sequence of set accesses (domain number): 0x5, 0x3, 0x4, 0x2, 0x3, 0x1, 0x2.

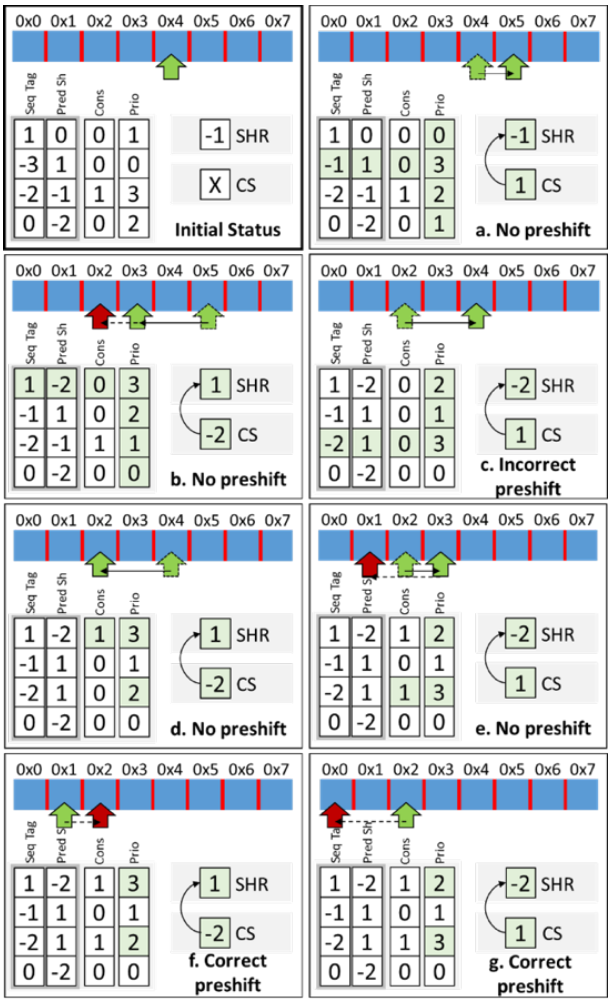

Fig. 4. Example

a. The head port was initially aligned to the previously accessed domain (0x4). As the first access is to 0x5 domain, the header must be shifted one position to the right (CS=1), adding an extra cycle to cache access. Concerning the Pattern Table update, the pattern [-1 1] resulting from the concatenation of the SHR (-1) and the Current Shift (1), is inserted in Pattern Table (SHR→Sequence Tag, Current Shift→Predicted Shift). The entry selected for the new pattern is the one with the lowest priority ([-3 1] in this case). SHR is updated with CS content (last step in Figure 3.b, turned arrow). Concerning Next shift prediction, a coincidence is found in the Sequence Tag (first row, value 1), but no preshift is done because this pattern is not consolidated remaining the head port aligned to the last accessed domain.

b. Next access moves the head port 2 positions to the left (CS=-2) to access domain 0x3. The SHR matches a Sequence tag of the table (first row, value 1) but Predicted Shift is different to Current Shift. Therefore, the new pattern [1 -2] (SHR + CS) replaces the pattern in the table with the same Sequence Tag. Next, SHR is updated to -2. In this case, there is a consolidated pattern matching Sequence Tag -2 so the head port is preshifted according to the predicted shift field (1 domain to the left, red arrow).

c. The speculative preshift turned out to be wrong, since the head was moved to the domain 0x2 but the new access refers to the domain 0x4, so the miss-prediction added one cycle to total latency. As a consequence, the consolidated pattern that triggered the preshift is replaced with the new pattern made up of SHR and Current Shift ([-2 -1] replaced with [-2 1]. Every replacement requires resetting the consolidation bit. Concerning prediction, there is a Sequence Tag with value 1 (matching the SHR updated value) but no preshift is done because it is not consolidated yet.

d. To access domain 0x2, the head port is shifted two positions to the left, pattern [1 -2] is consolidated (Sequence Tag and Predicted Shift match). No preshift is done (no consolidation of [-2 1] pattern).

e. The head port is shifted one position to the right and pattern [-2 1] is consolidated. The head port is preshitfted two positions to the left according to the consolidated pattern [1 -2], aligning the head port to domain 0x1.

f. In this case the preshift was correct and the head port is properly aligned when the new access arrives, eliminating the variable part of the access latency. Only priority bits are updated in the table. After updating the SHR value (-2), a match with the consolidated pattern [-2 1] is found and another preshift is done.

g. The preshift was correct again eliminating the need of shifting the head port. As in the previous access only LRU bits are updated in the table. After the SHR update (1) the head port is preshifted to domain 0x0 according to the consolidated pattern [1 -2].

IV. Evaluation Framework

We use Gem5 [27] as the main tool for our evaluation, modeling full-system activity. We simulate a 4-core CMP with the configuration parameters provided in TABLE I. .

TABLE I. Core and Cache Hierarchy configuration

Core Arch.	Functional Units	4×I-ALU/4×FP-ALU/4×D-MEM
	ROB size/Issue Width	128, 4-way
	Frequency/Count	3Ghz, 4 core
Private Caches	(L1)Size/Associativity / Block Size / Access Time	32KB I/ 32KB D (128KB RM), 4-way, 64B, 2 cycle
	(L2)Size / Associativity/ Block Size / Access Time/Type	256KB (4MB RM) Unified, 8-way, 64B, 10 cycles, Exclusive with L1
Shared L3	Size / Associativity / Block Size/Type	16MB (64MB RM), 16-way, 64B, Mostly inclusive / 24 cycles
	Coherence Prot., Consistency Mod.	MOESI snooping / TSO / Tagged prefetcher
RM	Domains / RW ports / Shift speed	64 / 1 rw port / 1cycle per domain
Mem.	Capacity / Access Time /BW	4GB /250 cycles / 32GB/s

58 diverse workloads, running on top of the Debian 8 OS, have been considered for evaluation. We make use of three of the most significant benchmark suites in computer architecture

to generate these workloads: SPEC2006 [28], PARSEC [29] and NAS Parallel Benchmark [30]. We also make use of the Yahoo! Cloud Serving Benchmark (YCSB) [31] as interface for four different database applications: Cassandra [32], MongoDB [33], OrientDB [34] and Redis [35]. The databases selected cover alternative NoSQL data models (column, graph and document-based) and performance optimizations (in-memory storage). The core package includes a set of pre-defined workloads (WA to WF) that try to model different scenarios [31]. TABLE II. summarizes the applications that form our evaluation framework.

TABLE II. WORKLOADS

SPEC CPU2006 (SINT, SFP)
astar, bzip2, gcc, gobmk, h264ref, hmmer, libquantum, mcf, sjeng, xalancbmk, bwaves, cactusADM, dealII, games, gromacs, lbm, milc, namd, soplex, sphinx3, zeusmp.
PARSEC 3.0 (PARSEC)
blackscholes, bodytrack, canneal, facesim, fluidanimate, raytrace
NPB 3.3.1 (NPB)
BT, CG, FT, IS, LU, MG, SP, UA
YCSB (CASS, MON, ORI, RED)
Cassandra (WA-WF), MongoDB (WA-WF), OrientDB (WA-WF), Redis (WA-WF).

Accurate hardware simulation is employed during the execution of the Region Of Interest (ROI). The ROI is reached making use of Virtual Machine (VM) based simulation acceleration [36]. Once reached, a checkpoint is taken and will be loaded subsequently in detailed architectural simulation. Starting from each checkpoint, the memory hierarchy is warmed up for enough cycles before starting to collect statistics, minimizing the effect of compulsory (cold) misses and warming-up non-architectural state (prefetchers, replacement policy, etc.). For each workload evaluated, multiple runs are employed to fulfill strict 95% confidence intervals.

V. PARAMETER SENSITIVITY

Next, we will analyze the impact of the configuration on the behavior of the proposed head-management policy. The main configuration parameters to study are: head tracking pattern type, pattern-length, consolidation threshold and table size. From these results, we will choose the actual values (which will definitively determine the feasibility of the idea itself). The results in this section show the process carried out to choose the appropriate configuration of the L2 cache. Similar analyses have been performed in each level of the cache-hierarchy.

A. Head Tracking

We evaluate two alternative options to track the header movement: relative to the current position (including distance and direction) and absolute (only considering the domain number). Tracking a pattern based on domain number implies that a cache access sequence must refer to the same domains at least twice to form a pattern (e.g., the following domain access sequence *0x4→0x7→0xC→0x4→0x7→0xC* would be recognized as the pattern [0x4 0x7 0xC]). In contrast, shift-based patterns track the head port movements instead of the domains accessed. This means that the identification of a shift-based pattern may not involve the same domains, only the same shift-sequence (e.g., the cache access sequence *0x1→0x2→0x4→0x7→0x8→0xA→0xD*, which requires the following head port movements *+1 +2 +3 +1 +2 +3*, would not be recognized as a domain pattern because the accesses refer to different domains, but it would be recognized as the shift pattern [1 2 3]).

In Fig. 5 we show the average shift values obtained for both approaches. Each column represents a different benchmark suite. The results are normalized to the ones obtained with domain-based patterns and the last column represents the geometric mean values. The preshifting policy clearly works better when shift-based patterns are employed. This is especially true for NPB suite and also for some benchmarks from SPEC (SINT and SFP). There are only three workloads (milc from SPEC (12%), canneal (1%) and facesim (2,5%) from PARSEC) which exhibit better performance working with domains. NoSQL applications (specially MON, RED and ORI) present a much lower sensitivity to this configuration parameter. In general, we observed that the number of patterns identified is lower when the policy works with domain-based patterns, moreover, the number of incorrect predictions is significantly higher.

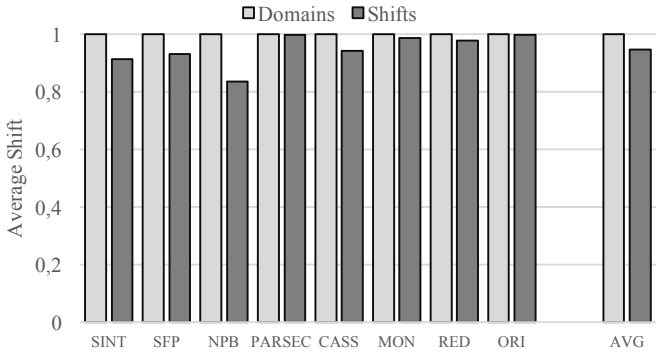

Fig. 5. Normalized average shift observed comparing shift-based tracking with domain-based tracking

In contrast, shift-based patterns are much more frequent and, moreover, the same pattern seems to be valid for different cache access sequences involving different domains (the distance between accesses remains for different addresses). These results guide our first design decision, shift-based preshifting being the choice for the remaining experiments of this paper.

B. Pattern Length

This might be the most critical parameter, because it affects prediction accuracy, prediction frequency and implementation cost. The trade-off between prediction accuracy and frequency must be analyzed. The use of short Pattern Length values harms accuracy, but longer patterns also reduce the frequency of pattern detection. Concerning implementation cost, short patterns are desirable, in order to reduce area and energy on Pattern Table implementation.

We analyze this parameter for four different values, increasing pattern length from 2 to 5. Fig. 6 shows the results obtained, evaluating both the accuracy and frequency of predictions (above) and the effect on average access latency (below). In both cases results have been normalized to those obtained for the shortest pattern. As expected, longer patterns improve prediction accuracy but reduce the number of preshifts (since it will be harder to have a hit on Pattern Table). For the longest pattern analyzed preshift operations are nearly halved,

but accuracy only improves by 20%. This imbalance has a direct effect on performance, and the smaller number of preshift operations degrades the average latency observed. The latency results indicate that short patterns are desirable, in order to maximize the number of predictions performed. As pattern length also affects Pattern Table size, we choose the shortest value (length=2) for the remaining experiments of this work.

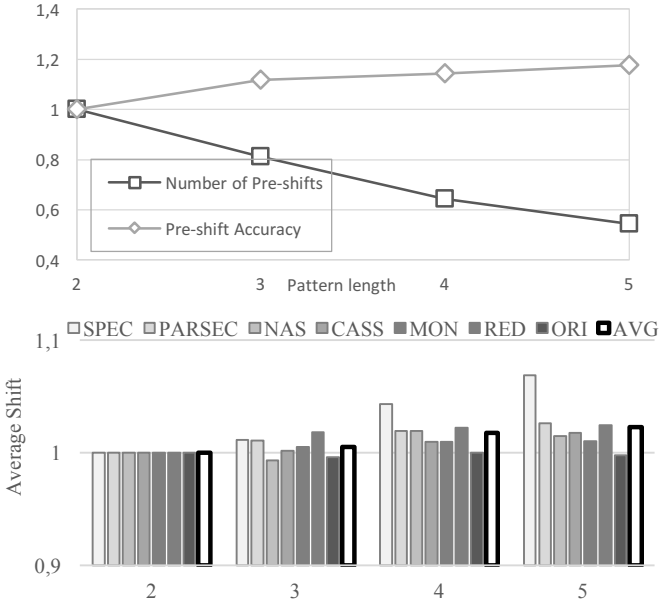

Fig. 6. (above) Preshift accuracy (correct preshifts) and frequency for different pattern lengths. (below) Normalized average shift observed for different pattern lengths.

C. Consolidation

The term consolidation refers to the number of repetitions of a pattern that are required to consider performing a preshift operation. Similarly to Pattern Length, this parameter has a direct influence on prediction accuracy and frequency. A larger consolidation value increases the probability of performing correct predictions, but also reduces the frequency of preshift operations.

Fig. 7 shows the sensitivity to consolidation. A value of 0 for consolidation means that the pattern is consolidated after only one occurrence. Whereas a value of 1 or higher for consolidation means that the pattern must appear at least twice to be consolidated. The first time it is inserted in the table, it being consolidated after the appropriate number of appearances is satisfied. Once consolidated, the next repetitions of the same pattern trigger a preshift operation. As can be seen, the most appropriate value for consolidation is 1. A value of 0 makes the policy very aggressive, degrading accuracy and increasing shift latency. Values larger than 1 make the policy to be conservative and also increase the average latency to access the cache bank. A noticeable degradation is observed when two or three repetitions are required for consolidation, especially for SPEC and PARSEC suites. Consequently, we will require a single pattern repetition before enabling preshift operations.

D. Pattern Table

The last parameter considered in this design space exploration is the size of the Pattern Table. Even for the smallest pattern length, storing all the possible pattern combinations would require a table with an unaffordable size. We implement a Pattern Table of limited size, limiting the number of patterns stored to only the last N detected (N represents the number of rows in the Pattern Table). In those cases where a replacement becomes necessary (an old pattern must be erased to include a new one), a LRU policy is implemented, similar to the one in charge of controlling cache-block replacement.

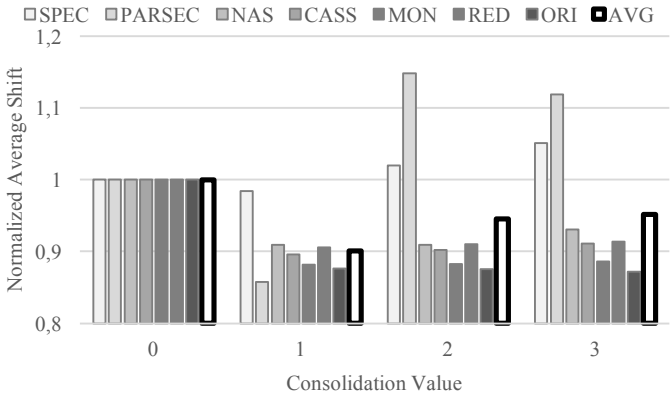

Fig. 7. Average shift for a growing consolidation threshold. Results normalized to the lowest consolidation value.

The experiment in Fig. 8 analyzes the capacity required by the proposed preshift policy. This analysis is done varying the number of entries in the Pattern Table, ranging from 4 to 128 entries. The results are again normalized to the ones with the smallest capacity (4 patterns stored). As expected, all benchmark suites undergo a shift latency decrease as the capacity increases but to a different degree. PARSEC reduces the average shift when varying the capacity from 4 to 128 entries by 5% approximately, SPEC goes further reducing it by 10%. The increase in capacity has little effect for NPB workloads, the average shift latency is only reduced by 2% approximately. Taking into account that the number of table entries is proportional to the cost, we have chosen a capacity value of 32, trying to balance performance and cost. In any case, it should be noticed that, if the cost is affordable, it is possible to increase the size of the Pattern Table to keep on improving performance.

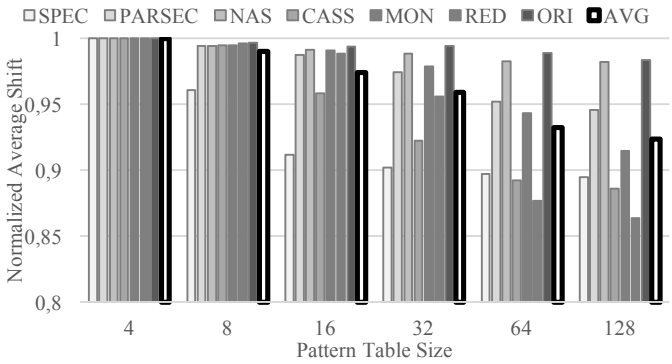

Fig. 8. Average shift evolution as the number of stored patterns increases. Results normalized to a single-pattern table.

E. Final Configuration & Cost Estimation

Each cache hierarchy level requires a similar set of experiments to figure out the most suitable configuration of the preshifting policy. For L1 and L2 levels, our mechanism implements the detection of shift-based patterns with a length of

2-elements, consolidated after the first repetition. The Table stores a total number of 32 patterns simultaneously, 16 in the case of L1. The Table size is increased to 64 patterns in LLC, while pattern length and consolidation maintain the same values as in the rest of cache levels. A RM with 64 domains requires 7 bits to encode each shift (two's-complement). Therefore, each Table Entry consists of 14 bits for the pattern, 4 bits for priority and one more for consolidation, making a total table size of 38 bytes. One single table is required per RM row ($64 \times 64 \times 8$ total bytes, or Domains × Block Size × Associativity) in the cache, which implies a storage overhead of 0.1159% for both L2 and LLC (in LLC the increased number of Table rows is compensated by the larger associativity).

TABLE III. ENERGY AND AREA ESTIMATION

	2MB RM Bank	Pattern Table
Area (mm^2)	0.2007	0.0058
Read Energy (nJ)	0.3506	0.0056
Write Energy (nJ)	0.1491	0.0032
Shift Energy (nJ)	0.2310	--
Leakage (mW)	207	7.12

We have modeled the hardware structure that implements the Pattern Table making use of DESTINY [37], a design space exploration tool for SRAM, eDRAM and Non-volatile technologies similar to CACTI [38] and NVSim [39]. We obtain area, read/write energy and leakage power values for both the Pattern Table and the associated cache bank. Despite not being in the critical cache access path, we assume a performance oriented implementation based on a CAM-like structure (fully associative cache), which provides a worst-case scenario in terms of energy and area. The value obtained are summarized in TABLE III. . On the RM cache side the energy per access, assuming an average shift value of 5 domains (chosen according to the results observed in the next section), increases to 1.5nJ. In contrast, the dynamic energy consumed by the Pattern Table on each cache access (hit) is 0.0088nJ, which means a 0.58% energy overhead per hit. It should be noted that misses only consume energy on the Cache side, so the energy overhead per access would be even lower. Finally, concerning area values we observe that the Pattern Table only requires a 2.45% area overhead to be implemented.

VI. PERFORMANCE EVALUATION

In this Section we compare the proposed mechanism (PRESHIFT) to the header management policies previously proposed. As well as Pattern-Preshift, the remaining results include the following counterparts:

- EAGER [13]: This policy restores the head port to a default location after each access. In the case of RM with a single read/write port the default position selected is the domain in the middle of the wire in order to reduce the worst-case shift latency.
- LAZY [13]: This policy leaves the head port in the position of the last access, trying to exploit both temporal and spatial locality. Shift penalty is zero when two or more consecutive accesses refer to the same domain.
- NEXT-BLOCK preshift [26]: The last policy performs static prediction. The head port is preshifted to the adjacent domain (+1) before the next access arrives.

A. Average shift operations at each cache level

The first set of results in this Performance Evaluation section is devoted to testing the latency overhead of a RM operating with each header management policy. To do so, we analyze the average shift length (variable part of RM access latency) required at each cache level to complete the execution of the workloads proposed in TABLE II. . The results are shown in Fig. 9, where the potential advantages of PRESHIFT over the other counterparts are summarized. To facilitate the interpretation of the results, they have been normalized to the ones obtained by LAZY policy and applications are ordered from best to worst performance.

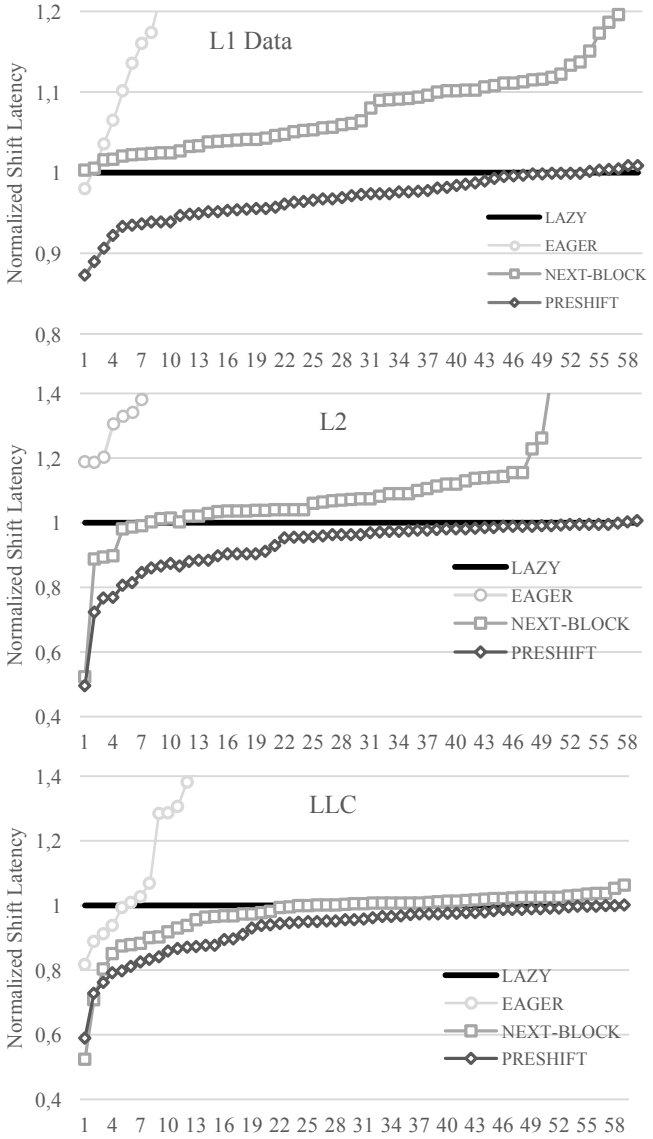

Fig. 9. Average shift latency for the policies evaluated. Results normalized to LAZY policy. One graph per cache level, workloads ordered from best to worst performance.

The results show a different behavior of the proposed counterparts at each cache level. In L1, NEXT-BLOCK preshift is not able to improve on the LAZY results for any workload. At this level there is a high degree of locality, which clearly benefits the LAZY policy. PRESHIFT preshift has a limited margin for

improvement and only eight workloads show a shift-length improvement over 5%. However, PRESHIFT preshift can improve the average shift length by up to 12% in a few cases, being slightly worse than LAZY in 6 workloads.

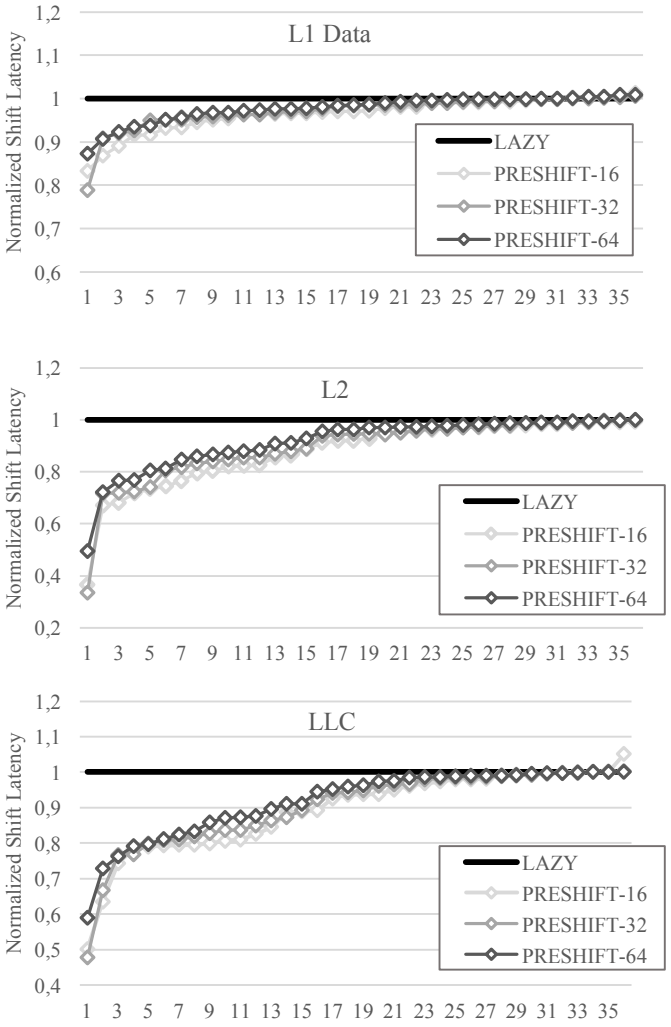

Fig. 10. Average shift latency for a RM with different number of domains. Results normalized to LAZY policy. One grapth per cache level, workloads ordered from best to worst performance.

Moving to L2 and LLC, part of the locality has been filtered, reducing the potential benefits of the LAZY policy. Additionally, the storage area is shared by data and instructions and in the case of the LLC, all cores within the chip make use of its banks simultaneously. For these reasons, the variability of cache accesses is increased, harming static policies (both LAZY and NEXT-BLOCK preshift). Comparing LAZY and NEXT-BLOCK, the shift latency clearly depends on the workload and its access pattern, it being hard to opt for any of the proposals. At L2 level LAZY seems to perform better than NEXT-BLOCK (only 7 workloads behave better with NEXT-BLOCK), but in LLC these results are much more balanced. Focusing on PRESHIFT preshift results at L2 and LLC caches, we observe how this policy outperforms its counterparts in both cases. In L2 cache half of the workloads improve on LAZY results by approximately 5%, reaching a shift-latency reduction up to 50%. In Last Level cache a similar behavior is observed. The only difference in this case is the significant improvement of NEXT-BLOCK preshift, which can achieve results close to PRESHIFT's. From this set of results we can conclude that the PRESHIFT preshift mechanism proposed can consistently outperform all of its counterparts, independently of the level in the cache hierarchy analyzed. The following section evaluates the effects of shift-latency reduction on system performance.

B. Number of Domains

The proposed evaluation process makes use of a fixed RM configuration, consisting of a racetrack with 64 domains and a single port. This set up tries to maximize storage density, one of the main advantages of the proposed technology. However, the literature provides examples where RMs are configured with a lower number of domains or multiple read/write ports. For this reason, we consider necessary the evaluation of the proposed prefetch policy for a variable number of domains. We conduct a similar evaluation to the one performed in previous section with different number of domains: 64, 32 and 16. Fig. 11 shows the results obtained for each cache level. We only include PRESHIFT results normalized to LAZY, being the counterpart with the closest performance. As can be seen, PRESHIFT improvement is consistent for every RM length analyzed. In fact, as the number of domains decreases, PRESHIFT improves its results. Our proposal is able to detect even more patterns in the presence of less domains, reducing the number of shifts required to reach the access port.

C. Overall memory latency

The previous experiments have demonstrated that the proposed policy can reduce the average shift latency when applied to any of the different cache levels. Thanks to the head port alignment policy, the variable part of the access latency can be significantly reduced. In this section we move one step forward toward measuring the impact of this reduction on the average memory latency. We employ multi-domain cells (64 domains) for both L2 and LLC. We discard the utilization of RM cells for the first level of cache. In this level, access latency is critical and shift latency harms performance significantly. As L1 hit latency has a fixed value, this metric does not provide relevant information about the performance of the different header management policies. For this reason, we choose the average memory observed to access the RM levels (equivalent to L1 miss latency) as a performance metric for the evaluated policies.

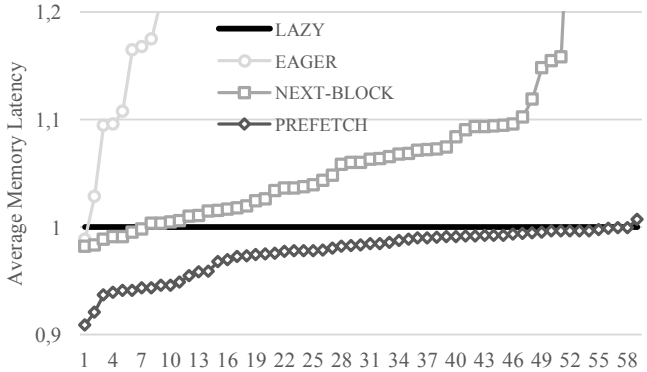

Fig. 11. Average latency observed to access the RM elements of the cache hierarchy. Results normalized to LAZY values.

In Fig. 11, we show the average latency observed to access the RM elements of the cache hierarchy. Main Memory latency is also taken into account to calculate these average latency results. We evaluate the same header management policies as in the previous section. Y-axis is again normalized to LAZY results, the best performer of all the counterparts. Applications are arranged according to PRESHIFT results, from best (left side) to worst (right side). The simulated system is configured according to the parameters inTABLE I. .

These results confirm the behavior observed in the previous section. EAGER and NEXT-BLOCK policies are unable to outperform LAZY latency for most of the applications evaluated. Despite NEXT-BLOCK showing a similar result to PRESHIFT in LLC (see Fig. 9), its poor results in L2 degrade the overall latency observed. In contrast, the consistent behavior of our proposal across the different cache levels leads better behavior of PRESHIFT results when compared to LAZY. For the set of applications analyzed, PRESHIFT obtains better latency results in 34 out of 35 workloads. This improvement exceeds 5% for 5 workloads, being close to 10% for the most favorable application.

A reduction in improvement margin is observed comparing Fig. 9 to Fig. 11 results. It should be noted that the achievable latency improvement is highly dependent on the combination of pattern location in both L2 and LLC levels, as well as on the miss rate observed in LLC (frequent Main memory accesses could hide the benefits of any header management policy). This means that in order to maximize the benefit of the PRESHIFT policy, both cache levels should present favorable predictions simultaneously and LLC misses should be minimal. In any case, for the set of workloads analyzed, the PRESHIFT policy never degrades latency results and we have been able to observe peak IPC improvement close to 5% for applications such as lbm.

VII. Conclusions

Domain Wall Memory is a promising storage technology that provides more density and requires less energy than other memory technologies such as SRAM or DRAM. The main drawback is the non-static access latency due the need of aligning the head port to the accessed one. We propose a new preshift header management policy that makes use of pattern-based prefetching mechanisms. We have carried out a detailed evaluation of the proposal, looking for its optimal parametrization and comparing it to state-of-the-art counterparts. Our results show that with minimal storage overhead, our proposal is able to obtain the shortest shift latency. We improve on previous policies in the literature by up to 50% in some cases. Additionally, our policy is as good as the best of the counterparts in those cases where it cannot take advantage of pattern detection. Finally, we have observed that the reduction in shift latency translates into a latency improvement up to 10% even in the set of workloads evaluated.

Acknowledgements

References

[1] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, “Clearing the Clouds: A Study of Emerging Scale-out Workloads on Modern Hardware,” in *ASPLOS’12*, 2012, vol. 40, no. Asplos, pp. 37–48.

[2] S. S. P. Parkin, M. Hayashi, and L. Thomas, “Magnetic Domain-Wall Racetrack Memory,” *Science (80-.).*, vol. 320, no. 5873, pp. 190–194, 2008.

[3] Y. Zhang, W. S. Zhao, D. Ravelosona, J. O. Klein, J. V. Kim, and C. Chappert, “Perpendicular-magnetic-anisotropy CoFeB racetrack memory,” *J. Appl. Phys.*, vol. 111, no. 9, 2012.

[4] L. Thomas, B. Hughes, C. Rettner, and S. S. P. Parkin, “Racetrack Memory: A high-performance, low-cost, non-volatile memory based on magnetic domain walls,” in *2011 International Electron Devices Meeting*, 2011, p. 24.2.1-24.2.4.

[5] W. Kang, Y. Huang, X. Zhang, Y. Zhou, and W. Zhao, “Skyrmion-Electronics: An Overview and Outlook,” *Proc. IEEE*, vol. 104, no. 10, pp. 2040–2061, 2016.

[6] M. Hosomi, H. Yamagishi, T. Yamamoto, K. Bessho, Y. Higo, K. Yamane, H. Yamada, M. Shoji, H. Hachino, C. Fukumoto, H. Nagao, and H. Kano, “A novel nonvolatile memory with spin torque transfer magnetization switching: spin-ram,” *IEEE Int. Devices Meet. 2005 IEDM Tech. Dig.*, vol. 0, no. c, pp. 459–462, 2005.

[7] B. C. Lee, P. Zhou, J. Yang, Y. Zhang, B. Zhao, E. Ipek, O. Mutlu, and D. Burger, “Phase-change technology and the future of main memory,” *IEEE Micro*, vol. 30, no. 1, pp. 131–141, 2010.

[8] R. Wiesendanger, “Nanoscale magnetic skyrmions in metallic films and multilayers: A new twist for spintronics,” *Nature Reviews Materials*, vol. 1, no. 7. 2016.

[9] G. Sun, C. Zhang, H. Li, Y. Zhang, W. Zhang, and Y. Gu, “From Device to System : Cross-layer Design Exploration of Racetrack Memory,” *Des. Autom. Test Eur. Conf. Exhib. (DATE), 2015*, pp. 1018–1023, 2015.

[10] M. Mao, W. Wen, Y. Zhang, Y. Chen, and H. (Helen) Li, “Exploration of GPGPU Register File Architecture Using Domain-wall-shift-write based Racetrack Memory,” in *Proceedings of the The 51st Annual Design Automation Conference on Design Automation Conference - DAC ’14*, 2014, pp. 1–6.

[11] R. Venkatesan, V. J. Kozhikkottu, M. Sharad, C. Augustine, A. Raychowdhury, K. Roy, and A. Raghunathan, “Cache Design with Domain Wall Memory,” *IEEE Trans. Comput.*, vol. 65, no. 4, pp. 1010–1024, 2016.

[12] H. Xu, Y. Alkabani, R. Melhem, and A. K. Jones, “FusedCache: A Naturally Inclusive, Racetrack Memory, Dual-Level Private Cache,” *IEEE Trans. Multi-Scale Comput. Syst.*, vol. 2, no. 2, pp. 69–82, 2016.

[13] 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,” *Proc. 2012 ACM/IEEE Int. Symp. Low power Electron. Des. - ISLPED ’12*, p. 185, 2012.

[14] S. Fukami, T. Suzuki, K. Nagahara, N. Ohshima, Y. Ozaki, S. Saito, R. Nebashi, N. Sakimura, H. Honjo, K. Mori, C. Igarashi, S. Miura, N. Ishiwata, and T. Sugibayashi, "Low-Current Perpendicular Domain Wall Motion Cell for Scalable High-Speed MRAM," *2009 Symp. VLSI Technol. Dig. Tech. Pap.*, pp. 230–231, 2009.

[15] A. J. Annunziata, M. C. Gaidis, L. Thomas, C. W. Chien, C. C. Hung, P. Chevalier, E. J. O'Sullivan, J. P. Hummel, E. A. Joseph, Y. Zhu, T. Topuria, E. Delenia, P. M. Rice, S. S. P. Parkin, and W. J. Gallagher, "Racetrack memory cell array with integrated magnetic tunnel junction readout," in *Technical Digest - International Electron Devices Meeting, IEDM*, 2011.

[16] S. Fukami, M. Yamanouchi, K. J. Kim, T. Suzuki, N. Sakimura, D. Chiba, S. Ikeda, T. Sugibayashi, N. Kasai, T. Ono, and H. Ohno, "20-nm magnetic domain wall motion memory with ultralow-power operation," in *Technical Digest - International Electron Devices Meeting, IEDM*, 2013.

[17] W. Kang, C. Zheng, Y. Huang, X. Zhang, Y. Zhou, W. Lv, and W. Zhao, "Complementary Skyrmion Racetrack Memory with Voltage Manipulation," *IEEE Electron Device Lett.*, vol. 37, no. 7, pp. 924–927, 2016.

[18] R. Tomasello, E. Martinez, R. Zivieri, L. Torres, M. Carpentieri, and G. Finocchio, "A strategy for the design of skyrmion racetrack memories," *Sci. Rep.*, vol. 4, p. 6784, Oct. 2014.

[19] D. Zhu, W. Kang, S. Li, Y. Huang, X. Zhang, Y. Zhou, and W. Zhao, "Skyrmion Racetrack Memory With Random Information Update/Deletion/Insertion," *IEEE Trans. Electron Devices*, vol. 65, no. 1, pp. 87–95, Jan. 2018.

[20] S. Mittal, "A Survey of Techniques for Architecting Processor Components Using Domain-Wall Memory," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 2, pp. 1–25, 2016.

[21] Z. Sun, X. Bi, A. K. Jones, and H. Li, "Design exploration of racetrack lower-level caches," in *Proceedings of the 2014 international symposium on Low power electronics and design - ISLPED '14*, 2014, pp. 263–266.

[22] Z. Sun, X. Bi, W. Wu, S. Yoo, and H. H. Li, "Array Organization and Data Management Exploration in Racetrack Memory," *IEEE Trans. Comput.*, vol. 65, no. 4, pp. 1041–1054, 2016.

[23] M. Moeng, H. Xu, R. Melhem, and A. K. Jones, "ContextPreRF: Enhancing the Performance and Energy of GPUs with Nonuniform Register Access," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 24, no. 1, pp. 343–347, 2016.

[24] E. Atoofian, "Reducing shift penalty in Domain Wall Memory through register locality," in *2015 International Conference on Compilers, Architecture and Synthesis for Embedded Systems, CASES 2015*, 2015, pp. 177–186.

[25] Y. Liang and S. Wang, "Performance-Centric Optimization for Racetrack Memory Based Register File on GPUs," *J. Comput. Sci. Technol.*, vol. 31, no. 1, pp. 36–49, 2016.

[26] R. Venkatesan, M. Sharad, K. Roy, and A. Raghunathan, "DWM-TAPESTRI - an energy efficient all-spin cache using domain wall shift based writes," *Proc. Conf. Des. Autom. Test Eur.*, pp. 1825–1830, 2013.

[27] N. Binkert, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, D. A. Wood, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, and T. Krishna, "The gem5 simulator," *ACM SIGARCH Comput. Archit. News*, vol. 39, no. 2, p. 1, 2011.

[28] SPEC Standard Performance Evaluation Corporation, "SPEC 2006," *https://spec.org*. .

[29] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques - PACT '08*, 2008, p. 72.

[30] H. Jin, M. Frumkin, and J. Yan, "The OpenMP implementation of NAS parallel benchmarks and its performance," *Natl. Aeronaut. Sp. Adm. (NASA), Tech. Rep. NAS-99-011, Moffett Field, USA*, no. October, 1999.

[31] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *Proceedings of the 1st ACM symposium on Cloud computing - SoCC '10*, 2010, p. 143.

[32] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, p. 35, Apr. 2010.

[33] "MongoDB." [Online]. Available: https://www.mongodb.com. [Accessed: 15-May-2017].

[34] O. T. LTD, "OrientDB," *Available: https://orientdb.com*. [Online]. Available: https://orientdb.com. [Accessed: 15-May-2017].

[35] S. Sanfilippo, "Redis," *Available: https://redis.io*. [Online]. Available: https://redis.io. [Accessed: 15-May-2017].

[36] S. Bischoff, A. Sandberg, A. Hansson, D. Sunwoo, A. G. Saidi, M. Horsnell, and B. M. Al-Hashimi, "Flexible and High-Speed System-Level Performance Analysis using Hardware-Accelerated Simulation," *Des. Autom. Test Eur.*, vol. 39, no. 2, p. 2012, 2013.

[37] S. Mittal, R. Wang, and J. Vetter, "DESTINY: A Comprehensive Tool with 3D and Multi-Level Cell Memory Modeling Capability," *J. Low Power Electron. Appl.*, vol. 7, no. 3, p. 23, 2017.

[38] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi, "Cacti 5.1," *HP Lab. April*, vol. 2, p. 24, 2008.

[39] N. P. Jouppi, "NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory," *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 31, no. 7, pp. 994–1007, Jul. 2012.