DyReCTape: A Dynamically Reconfigurable Cache using Domain Wall Memory Tapes

Ashish Ranjan, Shankar Ganesh Ramasubramanian, Rangharajan Venkatesan,
Vijay Pai, Kaushik Roy and Anand Raghunathan
School of Electrical and Computer Engineering, Purdue University
{aranjan,sramasub,rvenkate,vpai,kaushik,raghunathan}@purdue.edu

***Abstract*—Spintronic memories offer superior density, non-volatility and ultra-low standby power compared to CMOS memories, and have consequently attracted great interest in the design of on-chip caches. Domain Wall Memory (DWM) is a spintronic memory technology with unparalleled density arising from a tape-like structure. However, such a structure involves serialized access to the bits stored in each bit-cell, resulting in increased access latency, and thereby degrading performance. Prior efforts address this challenge either by limiting the number of bits per tape, in effect sacrificing the density benefits of DWM, or through cache management policies that can only partly alleviate the shift overhead.**

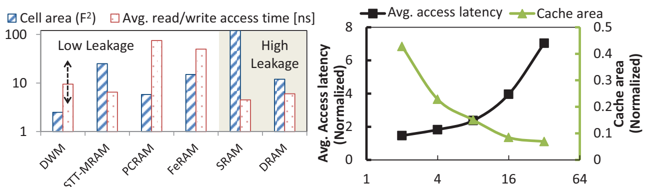
We observe that there exists significant heterogeneity in sensitivity to cache capacity and access latency across different applications, and across distinct phases of an application. We also make the key observation that DWM tapes offer a natural mechanism to tradeoff density for access latency by limiting the number of domains of each tape that are actively used to store cache data. Based on this insight, we propose DyReCTape, a dynamically reconfigurable cache that packs maximum bits per tape and leverages the intrinsic capability of DWMs to modulate the active bits per tape with minimal overhead. DyReCTape uses a history-based reconfiguration policy that tracks the number of shift operations incurred and miss rate to appropriately tailor the capacity and access latency of the DWM cache. We further propose two performance optimizations to DyReCTape: (i) a lazy migration policy to mitigate the overheads of reconfiguration, and (ii) re-use of the portion of the cache that is unused (due to reconfiguration) as a victim cache to reduce the number of off-chip accesses. We evaluate DyReCTape using applications from the PARSEC and SPLASH benchmark suites. Our experiments demonstrate that DyReCTape achieves 19.8% performance improvement over an iso-area SRAM cache and 11.7% performance improvement (due to a 3.4X reduction in the number of shifts) over a state-of-the-art DWM cache.

***Index Terms*—Spintronics, Domain Wall Memory, Racetrack Memory, Cache Design, Reconfigurable Caches**

I. Introduction

In the past several decades, the integrated circuit industry has witnessed a continuous surge in the demand for on-chip memory due to increasing data-set sizes and the widening processor-memory gap. A growing portion of chip area and energy consumption is expended in caches. Consequently, emerging memory technologies such as spintronic memories that promise high density and low leakage power are of great interest in cache design.

Domain Wall Memory (DWM) is a spintronic memory technology that achieves very high density and improved energy efficiency compared to CMOS and other emerging memory technologies, as shown in Fig. 1(a) [1], [2]. This has kindled great interest in using DWMs to realize caches both in the context of general purpose processors [3]–[7] and domain specific accelerators such as GPUs [8]–[10]. DWMs have a unique tape-like structure that achieves very high density by packing several (∼10s-100s of) bits into the domains of a ferromagnetic nanowire [1]. However, this structure also leads to serialized accesses to the bits in each bit-cell via shift operations, resulting in higher access latency. Fig. 1(b) shows the trade-off between cache area and average access latency with increasing bits per tape for PARSEC and SPLASH benchmarks. As the number of bits per tape increases, we observe a significant reduction in cache area. However, this also increases the number of shift operations, resulting in increased average access latency. This increased access latency for larger number of bits per tape is a major challenge in the design of DWM-based caches.



(a) Comparison of emerging memory technologies [1]

(b) Average latency *vs.* area trade-off

Fig. 1: DWM characteristics

In this work, we propose DyReCTape, a reconfigurable DWM-based cache that maximally exploits the density benefits of DWM while reducing the performance impact arising from shift operations. The DWM structure provides a natural mechanism to reconfigure cache size by varying the active number of bits per tape. DyReCTape leverages this unique capability of DWMs to dynamically modulate cache capacity and latency in order to improve overall system performance. In doing so, it considers the varying cache access patterns and sensitivity of applications to cache size and latency.

The design of DyReCTape poses two challenges: (i) the reconfiguration mechanism should track the performance sensitivity of applications to varying cache size as well as shift latency, and should respond to the dynamically changing memory access patterns within and across applications, and (ii) transitioning across different configurations of varying cache sizes involves data migration that leads to considerable energy and performance overheads. We address these challenges through suitable optimizations in DyReCTape.

In summary, the key contributions of this work are as follows:

- We propose the concept of a dynamically reconfigurable DWM-based cache that mitigates the performance penalty from shifts by modulating the bits per tape based on workload characteristics.
- We present a reconfigurable cache architecture consisting of (i) an address remapping scheme that seamlessly supports different logical-to-physical mappings of cache blocks for various cache configurations, and (ii) a history-based reconfiguration policy that dynamically tracks cache statistics such as the incurred penalties due to shifts and misses, and based thereupon, adapts

This work was supported by STAR-net, a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

cache capacity to the varying requirements within/across applications.

- We propose two key optimizations to further improve the performance of DyReCTape: (i) In order to minimize the overheads associated with data migration during reconfiguration, we propose a lazy migration policy by leveraging the non-volatility of DWMs. (ii) In order to further boost the performance, we utilize the inactive portion of cache after reconfiguration as a victim cache.
- We perform a detailed evaluation of DyReCTape using a device-to-architecture modeling framework. Across benchmarks from the SPLASH and PARSEC benchmark suites, DyReCTape achieves a 19.8% and 11.7% improvement in performance on average over a traditional SRAM cache and a static DWM-based cache, respectively.

The rest of the paper is organized as follows. Section II presents the necessary background information regarding domain wall memory and the bit cells used to realize the cache hierarchy. Section III describes the DyReCTape architecture along with various optimizations explored to further improve performance. Section IV explains the experimental methodology used to evaluate DyReCTape, and the results are presented in Section V. Section VI provides an overview of related previous work, and Section VII concludes the paper.

II. Background

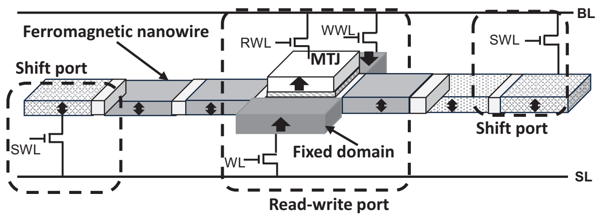


Fig. 2: Multi-bit DWM cell

Fig. 2 shows the structure of a multi-bit DWM cell consisting of a ferromagnetic nanowire, a read/write port, and shift ports. The ferromagnetic wire consists of multiple free domains, each of which can be programmed to store a bit. The read/write port is made up of a magnetic tunneling junction (MTJ), two fixed domains, and 3 access transistors. The MTJ is formed by a free domain of the nanowire and a fixed ferromagnet, separated by a tunneling oxide, and is used to sense the data stored in the free domain during the read operation. The fixed domains have opposite spin polarizations and are used to write to the free domain between them by shifting in the appropriate direction. The two access transistors at the extrema of the nanowire constitute the shift ports, which are used to inject a current pulse for moving the domains, thereby shifting the bits along the nanowire. Note that, a few extra domains on either end are reserved in order to ensure that there is no data loss as a result of shift operations.

Fig. 3 shows a logical view of the multi-bit DWM cell with the ferromagnetic nanowire represented as a tape storing multiple bits and the read-write port shown as a tape head. In order to access a bit stored in the tape, we need to shift the head to the desired location on the tape via the shift controller and then perform the required operation. Therefore, the access latency for a given bit stored in such a structure is variable and depends on the number of shift operations performed. In fact, packing a larger number of bits in the tape leads to an increase in the number of shifts, resulting in higher access latencies. On the other hand, storing a smaller number of bits reduces the access latency at the cost of reduced cache capacity. DWM, therefore, presents an interesting trade-off between latency and capacity. Our experiments for SPLASH and PARSEC benchmarks based on the setup described in section IV reveal that the performance penalty due to shifts outweighs the benefits from increased capacity, with the net penalty as high as 37%. In the next section, we describe a DWM-based cache architecture that exploits this trade-off to reconfigure the cache depending on the application's demand.

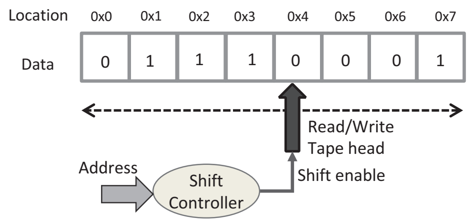


Fig. 3: Logical view of a multi-bit DWM cell

III. DyReCTape Architecture

Fig. 4 shows the overall architecture of DyReCTape. We follow the basic DWM-based cache organization described in [3], [4] — the tag array is designed with 1-bit DWM cells to avoid variable latency tag lookups, and the multi-bit DWM-based data array is organized into randomly addressable tape clusters [3]. We assume a bit-interleaved mapping of cache blocks to each tape cluster, such that a cache block can be accessed in parallel after the appropriate number of shift operations is applied to all the tapes in a tape cluster. We refer the interested reader to [3], [4] for further details, and focus on the architectural enhancements for reconfiguration.

For reconfiguration, DyReCTape also includes: (i) *address remapping logic* that supports different logical-to-physical mappings for various cache configurations, (ii) *data migration logic* that controls the movement of cache blocks while transitioning across multiple cache configurations, and (iii) a *reconfiguration controller*, which uses cache statistics such as number of shifts incurred and miss rate to initiate cache reconfiguration. In the following subsections, we provide a detailed description of the DyReCTape architecture and different optimizations explored to maximize performance.

A. Reconfiguration mechanism

DyReCTape involves dynamically varying the cache capacity by modulating the bits per tape. Let us consider an example to demonstrate how the reconfiguration is achieved. Fig. 5(a) shows a 32 entry direct-mapped cache that consists of 4 tape clusters with each cluster storing 8 cache blocks (8 bits/tape). In order to access a cache block in this design, the index bits are decomposed into decode and shift bits. The decode bits are used to select the tape cluster to which a cache block is mapped, and the shift bits are used to identify the location of the cache block within the tape cluster. For this example, we need 2 decode bits for identifying the cluster number and 3 shift bits for accessing the cache block within a cluster, which is also shown in Fig. 5(a). Suppose we wish to reduce the cache capacity by half, *i.e.*, reconfigure the cache as a 16 entry direct-mapped cache, we reduce the number of cache blocks stored per tape cluster to 4 (4 bits/tape). Fig. 5(b) shows the reconfigured cache state with the inactive portion of the cache highlighted in gray. We choose such a reconfiguration strategy because: (i) it lowers the number of shifts thereby improving cache latency, and (ii) it retains the spatial locality associated with data blocks residing in the active region of cache. We now require 2 decode bits and 2

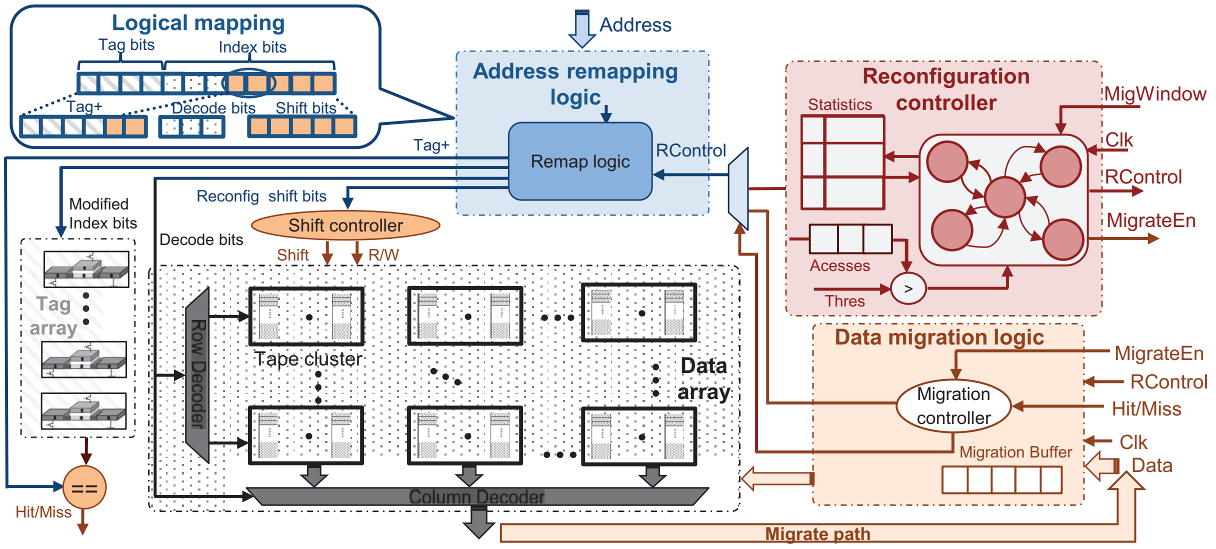


Fig. 4: DyReCTape organization

shift bits for accessing a cache block in the reconfigured state. Furthermore, determining hits/misses would require a wider tag (Tag+) which is composed of the original tag bits and the MSB of the original shift bits. In summary, it is necessary to have a suitable address remapping scheme in order to access a cache block from different reconfigured states.

Address remapping logic: In this work, we devise an addressing mechanism that can seamlessly support the different logical-to-physical mappings of cache blocks that manifest under various cache capacity states in a reconfigurable cache. This logical mapping is shown in Fig. 4. The address remapping logic automatically performs such a translation of an input address depending on the current cache configuration. As illustrated in Fig. 4, the inputs to the address remapping logic are the address for the cache block to be accessed and a control signal indicating the current cache configuration. It produces the modified index bits used to access the tag array, the decode bits for choosing the tape cluster and the modified shift bits, used to access the cache block in the reconfigured tape cluster. In this addressing scheme, we design the tag array to support the widest tag possible, as determined by the smallest cache capacity that is supported by DyReCTape, and use it to identify hits/misses for all possible cache sizes. Note that, the area and energy overheads to enable a wider tag array for our implementation are negligible and the redundant comparisons do not impact the correctness of tag lookups.

B. Reconfiguration policy

In this section, we describe the policy used to dynamically reconfigure the cache capacity in DyReCTape. Our reconfiguration policy considers the following cache statistics: (i) effective number of shifts, (ii) shift latency, (iii) miss latency, and (iv) effective miss rate. We divide the program execution into intervals that contain equal numbers of memory accesses, and compute the cumulative shift and miss penalties in each interval in order to determine reconfiguration actions, *i.e.*, expand or shrink. Equation 1 and 2 shows the shift and miss penalties in the i^{th} interval of program execution.

$$CSP_i = \sum_j H_{ij} * N_{S_{ij}} * T_{shift} \qquad (1)$$

$$CMP_i = \sum_j (1 - H_{ij}) * T_{main} \qquad (2)$$

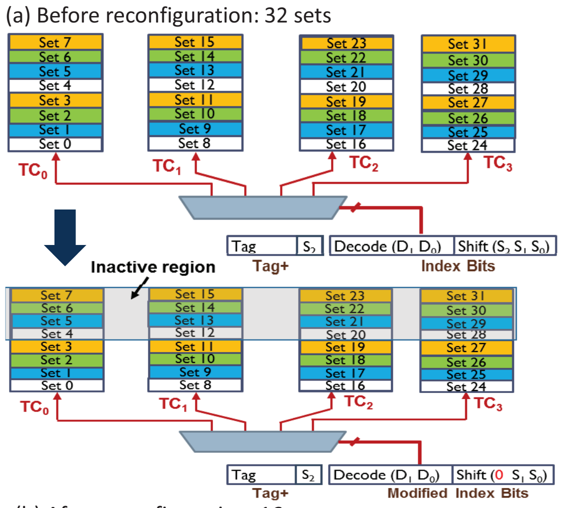
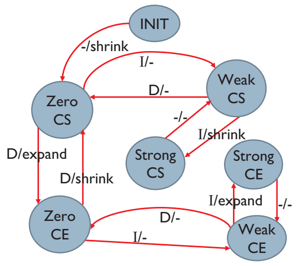


Fig. 5: Reconfiguration overview

In these equations, CSP_i denotes the cumulative shift penalty due to shift operations in the i^{th} interval. H_{ij} is a Boolean variable that indicates if the j^{th} access in interval i is a hit/miss, $N_{S_{ij}}$ is the numbers of shifts incurred during the j^{th} access and T_{shift} is the time required to shift the tape by one position. CMP_i represents the cumulative miss penalty in the i^{th} interval and T_{main} is the main memory latency. We compare the shift and miss penalties with a moving window average of their history to determine whether they are increasing or decreasing.

Fig. 6 illustrates the reconfiguration policy, which is based on the increasing/decreasing trend of CSP and CMP. Suppose that the prior reconfiguration action was an expand. If the increase in shift penalty outweighs the benefits of lower miss penalty in the current interval, we infer it as a degradation in cache performance and revert the expand action. On the other hand, if the decrease in cumulative miss penalty outweighs the corresponding shift penalty, the expand action is repeated. The conditions for improvement and degradation (shown in the table of Fig. 6) are exactly the opposite for a prior shrink operation. Note that, due to time-varying memory access characteristics, we might occasionally observe an effective increase or decrease in both shift and miss penalties. We therefore introduce thresholds (α, β) to determine the dominant factor amongst the two and use them to guide reconfiguration actions. In scenarios where we cannot determine the dominating penalty, the prior configuration is retained. In order to avoid frequent reconfiguration actions, we introduce three confidence states associated with every increase/decrease action, *viz.* zero, weak and strong. Depending on the history of improvements/degradation over their previous average values, we transition into one of the three states, *i.e.*, improvement leads to an increase in confidence and degradation lowers the confidence in our current reconfiguration action. Note that, reconfiguration actions are only performed in the zero confidence or strong confidence states. Zero confidence states (Zero CE and CS) cause an increase/decrease of cache size without any prior confidence while strong confidence states (Strong CE and CS) result in reconfiguration actions based on prior history of improvements. The actions are reversed only in the zero confidence states when we observe a degradation in cache performance.



Expand (Shrink)	(CSP$_i$ – Avg. CSP) ≥ α*(CMP$_i$ - Avg. CMP)	D (I)
	(CSP$_i$ – Avg. CSP) ≤ β*(CMP$_i$ - Avg. CMP)	I (D)

Zero confidence in shrink	Zero CS
Zero confidence in expand	Zero CE
Weak confidence in shrink	Weak CS
Weak confidence in expand	Weak CE
Strong confidence in shrink	Strong CS
Weak confidence in expand	Strong CE
I ➔ Improvement D ➔ Degradation	

Fig. 6: Reconfiguration policy

C. Data migration logic

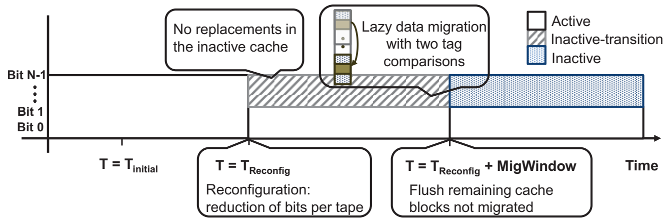
Reconfiguration involves an increase or decrease in cache capacity, which imposes the need for migration of cache blocks to ensure that they are stored at valid locations. We next discuss the different steps involved in this process and the proposed optimizations to reduce the associated overheads.

In DyReCTape, reconfiguration results in two kinds of actions: (i) shrink or decrease in bits per tape, and (ii) expand or increase in bits per tape. Shrink reduces the cache capacity, thereby rendering a portion of the cache inactive. Suitable actions are needed to handle the blocks that are present in the inactive portion. A naive solution for this would require flushing all the dirty cache blocks in the inactive portion to memory and subsequently invalidating them. On the other hand, those blocks present in the active portion require no action since their logical mapping remains valid. This naive scheme presents the following overheads: (i) heavy data traffic to main memory while flushing the cache blocks, and (ii) increase in miss rate due to the invalidated data.

Next when we consider the expand operation, it involves an increase in cache capacity, thereby transitioning a portion of the cache from inactive to active state. Therefore, a block that was previously stored at one location in the cache can get mapped to a different location in the newly activated region of the cache. Addressing this issue would require stalling all incoming cache accesses and pro-actively migrating all the cache blocks to their respective valid locations in the expanded cache. This unavailability of the cache for a large number of execution cycles leads to a significant performance overhead.

Our experiments confirm that these overheads are indeed significant enough to overshadow the performance improvements obtained as a result of reconfiguration.

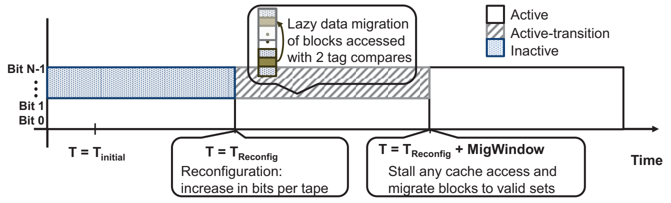
In order to mitigate the performance overheads associated with shrink and expand operations, we propose the concept of *lazy data migration* wherein data blocks are migrated over a migration time window to the appropriate location after reconfiguration. We next explain this scheme in detail for each of the two reconfiguration steps, *i.e.*, expand and shrink.



No replacements in the inactive cache
Lazy data migration with two tag comparisons
Active
Inactive-transition
Inactive
Bit N-1
Bit 1
Bit 0
T = T_initial
T = T_Reconfig
Reconfiguration: reduction of bits per tape
T = T_Reconfig + MigWindow
Flush remaining cache blocks not migrated
Time

Fig. 7: Timeline for reconfiguration during cache shrink

Lazy shrink: Fig. 7 demonstrates the lazy shrink policy with a timeline showing the decrease in bits/tape (*i.e.* capacity) over the migration window (denoted *MigrationWindow*). Before reconfiguration (at T = $T_{initial}$), the cache utilizes N bits per tape. At T = $T_{Reconfig}$, the reconfiguration action is initiated with a reduction in bits per tape. This renders a part of the cache inactive for normal cache operations. We note that the data already present in these locations can be retained without any overhead due to the non-volatile nature of DWMs. Based on this observation, the lazy shrink policy does not immediately flush the cache blocks in the inactive region after reconfiguration. Instead, the cache transitions lazily into the new reconfigured state over the *MigrationWindow* (see region marked Inactive-transition in Fig. 7). During the *MigrationWindow*, we perform the following actions upon each cache access: (i) We check for a hit/miss in the active region, (ii) If this results in a miss, the location in the inactive-transition region where the cache block would have resided before reconfiguration is checked. If the cache block is present in the inactive-transition region, we migrate it to the corresponding location in the active region. If the cache block is not present in either region, we declare a cache miss. At T = $T_{Reconfig} + MigrationWindow$, the few remaining dirty blocks in the inactive-transition region are flushed to main memory and the entire region is invalidated. In this way, the lazy shrink policy mitigates the performance overhead associated with migrating a large number of blocks at T = $T_{Reconfig}$ at the cost of marginally higher hit latency during the *MigrationWindow*. Our experiments suggest that the number of cache blocks that needs to be flushed at T = $T_{Reconfig} + MigrationWindow$ are much smaller ($\sim$5%) compared to the total dirty blocks at T = $T_{Reconfig}$.



Lazy data migration of blocks accessed with 2 tag compares
Active
Active-transition
Inactive
Bit N-1
Bit 1
Bit 0
T = T_initial
T = T_Reconfig
Reconfiguration: increase in bits per tape
T = T_Reconfig + MigWindow
Stall any cache access and migrate blocks to valid sets
Time

Fig. 8: Timeline for reconfiguration during cache expand

Lazy expand: Fig. 8 illustrates the lazy expand policy for increasing the cache capacity. In this case, the newly activated portion of the cache is marked as active-transition. Subsequently, for cache accesses within the *MigrationWindow*, the blocks can be found at a location either in the active or active-transition region. Therefore, we perform two tag comparisons in the same sequence as lazy shrink, and migrate the data to

TABLE I: System configuration

Processor Core	Alpha, out-of-order processor, 4 cores at 2 GHz
L1 I/D-cache	16KB per core, 2 way-set associative, 64B line size
L2 unified cache	2MB shared, 16 way-set associative, 64B line size
Cache latency	L1 cache: 2-cycle, L2 cache: 11-cycle
Main memory	2GB, 200-cycle

its correct location (if required) during the *MigrationWindow*. At $T = T_{Reconfig} + MigrationWindow$, we stall any further cache accesses and transfer the few remaining data blocks that require migration from the active region to their valid locations in the active-transition region. In this way, the performance overheads due to data migration are significantly lowered.

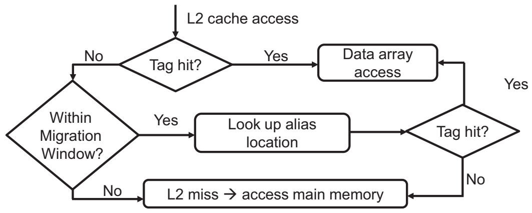


Fig. 9: Flowchart for overall L2 access

Fig. 9 summarizes the different steps involved in a single cache access upon reconfiguration. The key point to note is that alias locations, *i.e.*, the location of cache blocks before reconfiguration, are checked only during the *MigrationWindow*. The other steps are similar to a routine cache access.

D. Victim cache

When the cache is operating at less than maximum capacity, we propose to use the inactive portion as a victim cache. This victim cache holds the evicted blocks and improves the effective miss rate through reduction in conflict misses, resulting in further improvement of system performance. This modifies the sequence of steps upon a cache access as follows: (i) Upon a miss, we first check the victim cache, (ii) if we find the block, we bring it into the active cache and hold any blocks evicted during the process in the victim, else we go to main memory.

IV. Experimental Methodology

In this section, we discuss the experimental setup used to evaluate DyReCTape. The DWM device was modeled using a self-consistent physics-based device simulation framework proposed in [11]. The device parameters were then used to obtain the read, write and shift latency (and energy) for a DWM-based cache using DWM-CACTI [3]. In our experiments, a multi-bit cell capable of storing a maximum of 64 bits with 1 read/write port is considered. We modified gem5 [12] to model the DyReCTape architecture and performed architectural simulations to evaluate it. Table I shows the baseline system configuration. The 2MB SRAM L2 cache is replaced at iso-area by DyReCTape with a maximum capacity of 64 MB. We chose a migration window of 10^7 cycles with a sampling interval of 2000 accesses for dynamically reconfiguring the L2 cache. We perform a full-system simulation in the regions of interest for caches for various multi-threaded application benchmarks from PARSEC [13] and SPLASH-2x [14]. We utilize the cache access traces along with the cache characteristics from DWM-CACTI for estimating the energy consumed by DyReCTape. We use CACTI [15] for the SRAM cache, and a modified CACTI [16] for STT-MRAM in order to estimate the energy and access time. All memory technologies considered for our evaluation are based on a 32nm technology node.

V. Experimental Results

This section presents the results of various experiments that demonstrate the benefits of DyReCTape.

A. Performance evaluation

Fig. 10 summarizes the improvements obtained in IPC (instructions per cycle) across a wide range of benchmarks with DyReCTape. In this design, we consider six different baseline L2 cache designs under iso-area conditions: (i) an SRAM cache, (ii) an STT-MRAM cache, (iii) a static DWM-based cache (static-DWM) with 64MB capacity, (iv) a reconfigurable DWM cache with a naive approach in which data at invalid locations are flushed to memory or migrated to the correct location as soon as the cache is reconfigured (naive-recon-DWM), (v) a reconfigurable DWM cache with no victim cache for inactive portion (no-victim-DWM), and (vi) an "ideal" DWM cache (zero-shft-DWM) which provides the maximum capacity and also incurs no shift penalty. The IPC used for comparing these different baselines is normalized to the static organization of DWM-based L2 with maximum capacity, *i.e.*, 64MB. On average, DyReCTape achieves 19.8% and 9% IPC improvement over the SRAM and STT-MRAM designs. This is primarily because of two factors: (a) higher cache capacity (32X and 8X *w.r.t* SRAM and STT-MRAM, respectively) due to the density of DWM, and (b) the reconfigurable cache operating at more optimal latency and capacity points for these benchmarks.

Across all benchmarks, DyReCTape obtains an 11.7% improvement (on average) in IPC over a static organization of DWM-based cache with 64MB capacity. This increase in performance is mainly because of our scheme that suitably reconfigures the cache size to operate at the capacity/latency optimal point for performance, thereby reducing the overall shift operations (3.4X on average). Fig. 10 also shows that a naive reconfiguration which incurs the full transition overheads hardly obtains any performance benefit (∼1%). This underscores the utility of the lazy migration mechanism, which minimizes these transition overheads.

We also observe a 10.1% improvement in IPC from Fig. 10 over all the benchmarks, when DyReCTape uses only the proposed reconfiguration scheme without utilizing the inactive capacity as victim cache. The inclusion of victim cache provides an additional benefit of 5.8% on average for a subset of benchmarks (ferret, x264, vips, lu-contig and lu-noncontig) with substantial evictions. Note that, for some of the benchmarks in this subset (ferret, vips, lu-contig and lu-noncontig), DyReCTape outperforms even the DWM cache with no shift overheads, due to the use of a victim cache that lowers the conflict misses.

Overall, DyReCTape achieves a performance within 3.5% of the ideal DWM-based cache with no shift overheads (zero-shft-DWM), demonstrating the effectiveness of the proposed reconfiguration scheme in minimizing shift operations by utilizing the appropriate cache capacity for each application.

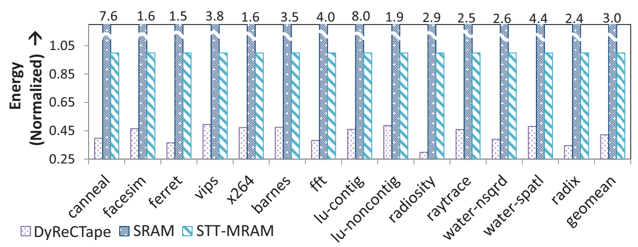


Fig. 11: Energy comparison for different baselines

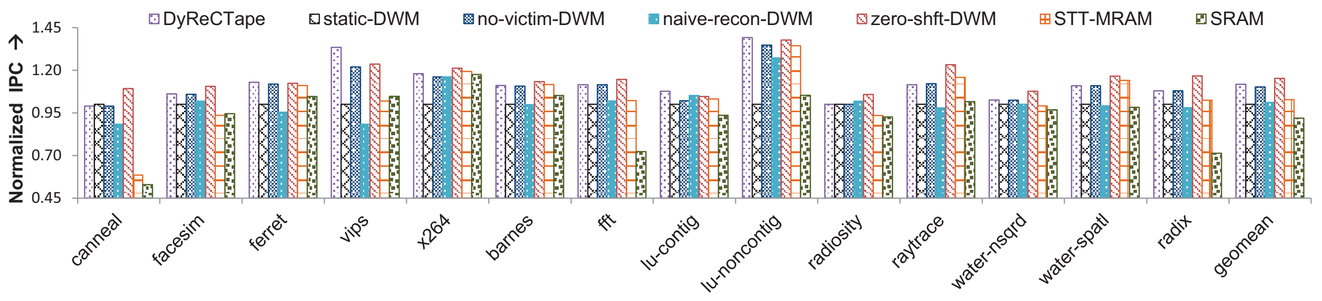


Fig. 10: Performance trends for different baselines

B. *Energy comparison*

Fig. 11 compares the energy consumed by DyReCTape with SRAM and STT-MRAM iso-area L2 caches. DyReCTape achieves a reduction of 7.1X in cache energy over SRAM. This is mainly due to the reduction in leakage energy because of the non-volatile nature of DWM. Further benefits are achieved due to techniques that lower their read/write energy [2]. In the case of STT-MRAM, we note an energy reduction of 2.4X primarily because of lower write energy of DWMs due to the previously proposed shift-based write mechanism [2].

In summary, the proposed reconfigurable cache is able to exploit the high density offered by DWMs while mitigating the shift penalty, and thereby achieve substantial benefits in both performance and energy.

VI. Related Work

In this section, we focus our discussion to prior work on DWM caches and reconfigurable caches.

DWM was first explored in the context of secondary storage [1] mainly due to its promising density. Considering its attractive benefits of both density and energy, researchers have investigated DWMs at the device and circuit level [4], [5], [11], [17], [18]. Multiple prototypes of DWM have also been demonstrated [19], [20]. Recent efforts [3]–[7], [9] have explored the applicability of DWM for on-chip memories. [3]–[6] explored DWM-based caches in the context of general purpose processors with fixed numbers of bits on a tape and proposed cache management policies to mitigate the performance penalty due to serialized memory access. Subsequently, [7] presented several layout strategies along with way-based mapping and resizing for DWM-based cache architecture. In addition, DWMs have also been explored in the context of graphics processors [9], [10] and accelerators [8].

Another area of related work is the design of reconfigurable caches. Reconfigurable caches have been extensively researched in the context of CMOS-based caches. Most of the efforts have targeted optimizing energy consumption with minimal impact on system performance. [21] proposed dynamically reconfiguring the associativity of caches, while [22] used associativity, capacity and line size reconfiguration for achieving energy efficiency. Few efforts [23], [24] have utilized dynamic reconfiguration of hybrid caches to address the write inefficiencies of STT-MRAMs.

Our work explores a dynamically reconfigurable architecture to address the unique challenge posed by DWM-based caches, *i.e.*, performance penalty due to shifts. The unique structure of DWM provides a natural knob to vary cache capacity and latency at run-time by modulating the bits per tape. However, as shown in our results, such a scheme also requires a suitable reconfiguration policy and optimizations that mitigate the reconfiguration overheads in order to achieve improved performance.

VII. Conclusion

Domain wall memory is a spintronic memory technology that provides superior density and energy efficiency compared to other emerging memory technologies. However, realizing larger capacity with such memories is challenging owing to the unique structure of DWM that requires shift operations for each access. The higher cache access latency due to shifts hurts overall system performance. In this work, we proposed DyReCTape, a DWM-based reconfigurable cache that dynamically tailors its cache size to improve performance. We propose optimizations that mitigate the overheads associated with reconfiguration. We performed architectural simulations on a wide range of benchmarks, and demonstrate substantial benefits in performance compared to a static DWM-based cache organization.

References

[1] S. Parkin et al. Magnetic domain-wall racetrack memory. *Science*, 2008.
[2] S. Fukami et al. Low-current perpendicular domain wall motion cell for scalable high-speed MRAM. In *Proc. VLSIT*, 2009.
[3] R. Venkatesan et al. TapeCache: A high density, energy efficient cache based on domain wall memory. In *Proc. ISLPED*, 2012.
[4] R. Venkatesan et al. DWM-TAPESTRI - An energy efficient all-spin cache using domain wall shift based writes. In *Proc. DATE*, 2013.
[5] Z. Sun et al. Cross-layer Racetrack Memory Design for Ultra High Density and Low Power Consumption. In *Proc. DAC*, 2013.
[6] S. Motaman et al. Synergistic circuit and system design for energy-efficient and robust domain wall caches. In *Proc. ISLPED*, 2014.
[7] Z. Sun et al. Design Exploration of Racetrack Lower-level Caches. In *Proc. ISLPED*, 2014.
[8] R. Venkatesan et. al. Domain-specific many-core computing using spin-based memory. *Nanotechnology, IEEE Transactions on*, Sept 2014.
[9] R. Venkatesan et al. STAG: Spintronic-Tape Architecture for GPGPU cache hierarchies. In *Proc. ISCA*, 2014.
[10] M. Mao et al. Exploration of GPGPU Register File Architecture Using Domain-wall-shift-write Based Racetrack Memory. In *Proc. DAC*, 2014.
[11] C. Augustine et al. Numerical analysis of domain wall propagation for dense memory arrays. In *Proc. IEDM*, 2011.
[12] N. Binkert et al. The gem5 simulator. *SIGARCH Compu. Arch. News'11*.
[13] C. Bienia et al. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *Proc. PACT*, 2008.
[14] SPLASH-2x. http://parsec.cs.princeton.edu/doc/memo−splash2x−input.pdf.
[15] CACTI, www.hpl.hp.com/research/cacti.
[16] X. Dong et al. Circuit and microarchitecture evaluation of 3D stacking magnetic RAM (MRAM) as a universal memory replacement. In *Proc. DAC*, June 2008.
[17] M. Sharad et al. Multi-level magnetic RAM using domain wall shift for energy-efficient, high-density caches. In *Proc. ISLPED*, 2013.
[18] A. Iyengar et al. Modeling and Analysis of Domain Wall Dynamics for Robust and Low-Power Embedded Memory. In *Proc. DAC*, 2014.
[19] A. J. Annunziata et al. Racetrack memory cell array with integrated magnetic tunnel junction readout. In *Proc. IEDM*, 2011.
[20] E. R. Lewis et al. Fast domain wall motion in magnetic comb structures. *Nature*, 2010.
[21] David H. Albonesi. Selective Cache Ways: On-demand Cache Resource Allocation. In *Proc. MICRO*, 1999.
[22] C. Zhang et. al. A highly configurable cache for low energy embedded systems. *ACM Trans. Embed. Comput. Syst.*, May 2005.
[23] J. Zhao et al. Bandwidth-aware reconfigurable cache design with hybrid memory technologies. In *Proc. ICCAD*, 2011.
[24] Y.T. Chen et al. Dynamically Reconfigurable Hybrid Cache: An Energy-efficient Last-level Cache Design. In *Proc. DATE*, 2012.