# BlendCache: An Energy and Area Efficient Racetrack Last-Level-Cache Architecture

Fazal Hameed and Jeronimo Castrillon, *Senior Member, IEEE*

***Abstract*—Racetrack memory (RTM) is a promising non-volatile memory that provides multi-bit storage cells achieving a higher area and leakage energy efficiency compared to contemporary volatile and non-volatile memories. These features make RTM a potential candidate to be used as a Last-Level-Cache (LLC). One drawback of the multi-bit RTM cell is the serialized access to the stored data, resulting in a shift penalty to access a particular bit within the cell. This overhead is particularly critical for LLC tags, for which prior RTM designs place tags either in SRAM or in single-bit RTM cells. While this avoids shifting, these designs require large number of leaky cells incurring high energy consumption. To address this problem, this paper proposes an energy efficient RTM design called BlendCache that efficiently stores the tags in the leakage optimized multi-bit RTM cells. To reduce the RTM shift penalty of these cells, BlendCache exploits the spatial locality of programs by maximizing accesses to nearby locations in RTM. Employing 32-bit RTM cells for a single-core, BlendCache reduces the energy consumption by 20.8% and area by 15.2% compared to the state-of-the-art while its impact on performance is negligible. For a 4-core system, the energy improvement translates to 35.9% with 3% performance degradation.**

***Index Terms*—Architecture, cache, embedded systems, memory, racetrack, shift.**

## I. Introduction

THE demand for larger Last-Level-Caches (LLC) has grown significantly due to widening processor-memory latency gap and large memory footprints of emerging applications. Prior research has adopted RTM as LLC due to its high area and energy efficiency compared to SRAM and Spin Transfer Torque (STT) memories [1]–[6]. These proposals split the LLC into separate tag and data arrays. The performance-critical tag array is implemented using latency optimized cells (i.e., SRAM [1], [2], [4], [5], STT [6] or single-bit RTM cells [3]). The data array, in turn, uses leakage-optimized multi-bit RTM cells. Although these designs significantly reduce the energy consumption compared to an iso-capacity SRAM cache, the leakage energy is dominated by the energy dissipated in the leaky storage cells dedicated for the tags. For instance, a small SRAM tag array of size 592 KiB is responsible for at least one-third of the overall energy consumption of an 8 MB RTM LLC (cf. Section IV-B and Fig. 10).

The multi-bit RTM cell achieves energy efficiency because the leaky access port (responsible for a read or a write operation) is shared by multiple locations (i.e., bits). These RTM cells require a shift operation to align the target bit location with the access port which introduces additional latency and energy penalties. Therefore, a common approach to reduce leakage is to employ single-port multi-bit RTM cells. The shift cost of the multi-bit cell can be alleviated by using more access ports. However, the area and leakage energy of the design grow with the number of ports, practically independent of the number of bits in the cell [1]–[3], [7]. Multi-port RTM cells are thus impractical not only due to leakage energy, as mentioned before, but also because they considerably reduce the memory density.

Several architectural solutions have been proposed to mitigate the negative impact of shift penalty in RTM caches. While these studies report a noticeable improvement in the shift energy, the leakage energy of the tag array has been neglected so far. For larger LLCs, this issue cannot be ignored anymore. Therefore, a natural approach to address the leakage energy problem is to use multi-bit cells for the tag array. Doing so requires to first shift the tags to determine whether an access will lead to an LLC hit or a miss. Additional shifts are then required in the RTM data array to access the data following an LLC hit. This double shift penalty would severely increase the overall LLC access latency.

In this paper, we propose a novel RTM LLC design that efficiently stores the tags in the multi-bit RTM cells. To reduce the shifting overhead of these multi-bit cells, our proposed RTM design exploits the program spatial locality by mapping spatial adjacent memory blocks to adjacent locations in the RTM which ensures that most of the cache accesses requires a small shift cost. Concretely, our contributions are:

1) An RTM tag store design that combines the tag and the cache line into the same index in multi-bit RTM cells. This reduces the leakage energy and allows the tags and the cache line to be accessed at the same time, thereby reducing the RTM access latency.
2) A direct-mapped organization in which the target RTM location is determined by the address bits directly. In contrast, state-of-the-art associative designs for RTM requires a tag lookup to determine the target RTM location.
3) A novel data mapping policy that assigns spatial adjacent memory blocks to consecutive RTM locations and exploits the program's spatial locality to reduce the RTM shift cost.
4) A detailed design space exploration of the impact of different RTM cell sizes on performance, area, and energy consumption.

The rest of the paper is organized as follows. First, we provide the background on RTM in Section II. The proposed RTM

The authors are with the Chair for Compiler Construction and the Center for Advancing Electronics Dresden, Technische Universität Dresden, 01069 Dresden, Germany.
E-mails: {fazal.hameed, jeronimo.castrillon}@tu-dresden.de
F. Hameed is also affiliated with the institute of space technologies (IST), Islamabad Pakistan.

tag store design, data mapping policy, the control flow, and the overhead analysis is discussed in Section III. Afterwards, the evaluation of the proposed RTM LLC with state-of-the-art designs is presented in Section IV. A summary of the related work is provided in Section V followed by the conclusion in Section VI.

## II. Background

This section provides a brief overview of RTM, discussing its cell structure, organization and the underlying cache architecture.

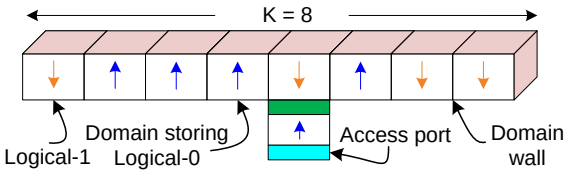


Fig. 1. A basic RTM cell (i.e., track) storing $K$ = 8 domain walls (i.e., bits)

### A. Basic RTM Cell

A single cell in the Racetrack memory (RTM) is referred to as a track which stores $K$ data bits in the form of magnetic domains as shown in Fig. 1 [8], [9]. The value of a domain is determined by its magnetic orientation indicated by direction of the arrow in Fig. 1. Each track is provided with an access port that enables read/write operation. Accessing a particular magnetic domain within a track requires to first align the target domain to the access port (referred to as *shift* operation) before performing a read or write operation. The shift cost to access a bit within a track vary from 0 (i.e., best case when access port is already aligned with the target bit) to $K$-1 (i.e., worst case).

The multi-bit RTM cell (i.e., $K > 1$) provides area and leakage energy efficiency compared to single-bit RTM cell (i.e., $K = 1$) but introduces daunting challenges in terms of latency/energy overheads, controller complexity and reliability. The leakage energy and area of RTM is primarily dominated by the access port (associated with the track) consisting of CMOS transistors. For a given capacity and fixed number of access ports per track, a higher $K$ reduces the total number of access ports compared to a smaller $K$. As a result, increasing $K$ provides better leakage and area efficiency but incurs high energy and latency per shift. Since increasing the number of access port per track negatively impacts the leakage energy, this work assumes a single access port per track. The access latency for a higher $K$ results in non-uniform access latency complicating RTM controller design. Furthermore, fluctuations in the shift current density may lead to position errors, thereby introducing reliability challenges [10], [11].

### B. Basic RTM Organization

A typical RTM organization (cf. Fig. 2) is divided into $B$ banks, where each bank is further subdivided into $S$ subarrays. Each subarray consists of $D$ domain block clusters (DBCs) [1], [3]. Each DBC is a group of $T$ tracks each storing $K$ domains. A $T$-bit data in a DBC is distributed in a bit-interleaved fashion across $T$ tracks which are accessed in parallel. Fig. 2 demonstrates how $K$ = 32 $T$-bit data are stored in $T$= 512 tracks. Each DBC maintains a track register to keep track of the current access port position (pointing to $L_r$ in Fig. 2) which points to one of the $K$ locations of the DBC. To access a particular data within a DBC, domains of all the $T$ tracks are shifted together in a lockstep fashion such that all the domains (or bits) of that data are aligned to their respective access port position. After that, all the bits of the relevant data are accessed simultaneously.

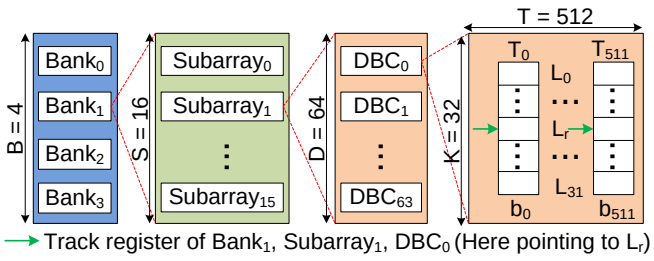


Fig. 2. A typical RTM organization. DBC stands for Domain Block Cluster

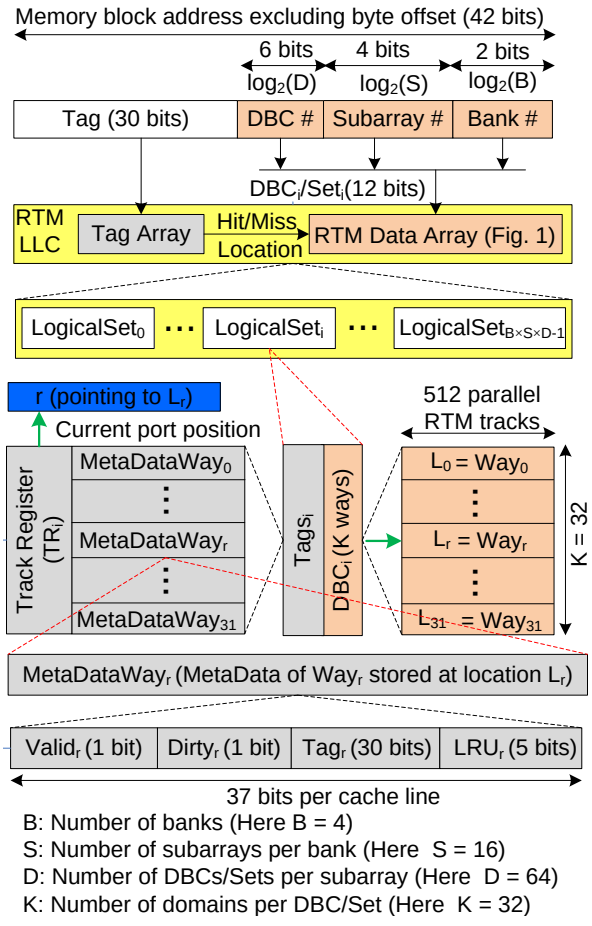


Fig. 3. The baseline RTM cache architecture

### C. RTM Cache Architecture

Traditional RTM cache designs implement the data array using RTM and the tag array using SRAM [1], [2], [4], [5] (cf. baseline architecture in Fig. 3). The RTM data array is hierarchically decomposed into banks, subarrays, DBCs, tracks and domains as described before. As shown in Fig. 3, the cache consists of $B \times S \times D$ logical sets. Each logical $Set_i$ ($i \in \{0, 1, .., B \times S \times D - 1\}$) consists of $Tags_i$ (i.e., tags of $Set_i$ stored in the SRAM tag array) and $DBC_i$. The *K* cache ways of $Set_i$ are stored in the RTM data array where each location $L_j$ ($j \in \{0, 1, .., K - 1\}$) of $DBC_i$ represents a way $Way_j$ of $Set_i$. Each track register ($TR_i$) of $log_2(K)$ bits points to one of the *K* ways in $Set_i$.

For illustration, we consider *B* equals to 4, *S* equals to 16, *D* equals to 64, *K* equals to 32, and *T* equals to 512. In this setting, the RTM cache stores 4096 logical sets with 32-way associativity each way storing a 512-bit cache line. The data mapping in the cache is shown on the top of Fig. 3. Excluding the byte offset bits of the address line, the lower 12 bits of the memory block address identify one of the DBCs (or logical sets) denoted by $DBC_i$. The exact location of the cache line $L_j$ within $Set_i$ is determined by a tag match (in case of a cache hit) or the least recently used cache line (in case of a cache miss). In this scenario, the shift cost depends on the absolute difference of the target cache line way number (i.e., *j*) and the current port position (denoted by *r* in Fig. 3) stored in $TR_i$.

## III. BlendCache LLC Architecture

This section provides the details of our novel RTM LLC architecture referred to as BlendCache. For energy efficiency, BlendCache stores the tags in the leakage optimized multi-bit RTM cells. Naively storing the tags in these cells may exacerbate the performance since an LLC hit would require two costly RTM accesses, one for the tag and other for the data. To address this problem, BlendCache stores the tag and the data into the same index. This allows the tag and data to be accessed at the same time in a single unified RTM access instead of two separate ones (cf. Section III-A). To provide fast LLC miss detection (cf. Section III-B) for critical read requests, BlendCache relies on an existing memory access predictor that quickly forwards the request to main memory. To improve the performance, we propose a novel data mapping policy that reduces the overall RTM shift cost compared to the traditional RTM LLC designs (cf. Section III-C).

### A. Tag Store Design

The baseline RTM LLC requires an SRAM tag lookup before accessing the cache line in the RTM data array as shown in Fig. 4-(a). To reduce the tag serialization latency caused by the SRAM tag access, BlendCache stores the tags and the cache line in the same location within a DBC as depicted in Fig. 5. A $DBC_i$ in the BlendCache is partitioned into *K* locations where *K* refers to the number of domains in a track (cf. Fig. 2). Each location $L_j$ ($j \in \{0, 1, .., K-1\}$) of $DBC_i$ stores $Line + MetaData$ bits where $Line$ (512 bits in Fig. 5) and $MetaData$ (27 bits in Fig. 5) are the cache line and tag

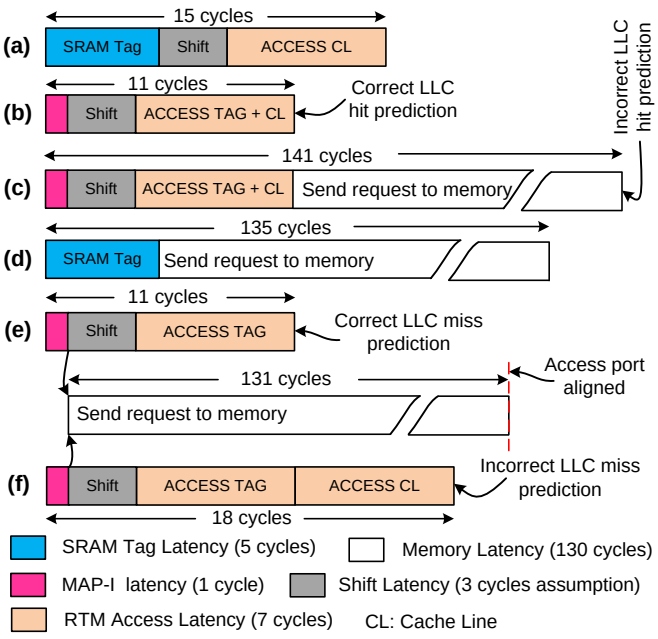


Fig. 4. Read latency breakdown for (a) LLC hit in the baseline LLC (b) LLC hit in BlendCache when MAP-I correctly predicts an LLC hit (c) LLC hit in BlendCache in case of wrong LLC hit prediction by MAP-I (d) LLC miss in the baseline LLC (e) LLC miss in BlendCache when MAP-I correctly predicts an LLC miss (f) LLC miss in BlendCache in case of wrong LLC miss prediction by MAP-I. Shift latency is assumed to be 3 cycles

entry size in bits, respectively. Each location $L_j$ of $DBC_i$ represents one set of the direct-mapped cache. An access to the BlendCache requires retrieving the relevant cache line at location $L_j$ which is identified by $log_2(K)$ of the memory block address bits (highlighted by red arrow in Fig. 5). An LLC hit occurs when the tag at location $L_j$ matches with the tag field of the memory block address. In this scenario, the corresponding cache line is returned to the core and the lower level caches.

### B. Fast LLC Miss Detection for Reads

Since BlendCache stores the tags in the RTM, the main challenge is to provide a fast and efficient LLC miss detection for critical reads to avoid waiting for the tag access. Ideally, an LLC read miss needs to be sent to main memory as soon as possible. However, detecting an LLC read miss in BlendCache requires a costly tag check in the RTM array which increases the LLC read miss latency. To address this problem, BlendCache relies on a low-overhead instruction-based memory access (MAP-I) predictor (MAP-I) [12] with a single-cycle latency. The MAP-I requires a storage overhead of 96 bytes and is implemented in SRAM which is employed to predict the outcome of an LLC read request (i.e., a hit or a miss). The LLC read miss latency in BlendCache is reduced by sending a request to memory before performing the costly tag check in RTM if the MAP-I predicts an LLC miss (cf. Fig. 4-e).

The control flow of the BlendCache is depicted in Fig. 6. Upon each LLC read request, BlendCache uses the MAP-I to predict an LLC hit or a miss. If the MAP-I predicts an LLC

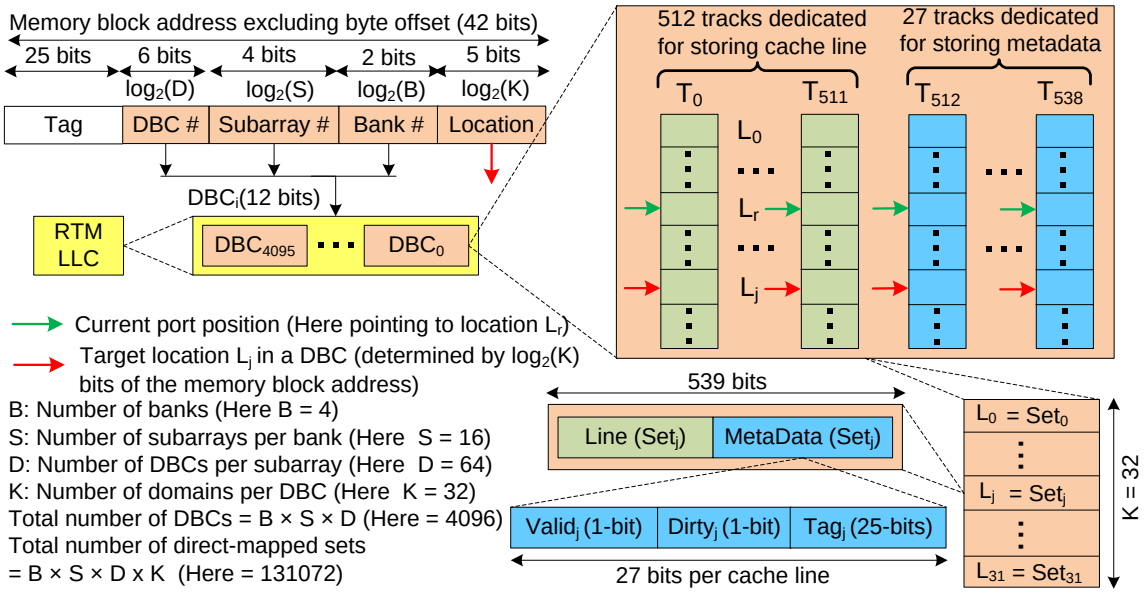


Fig. 5. BlendCache architecture

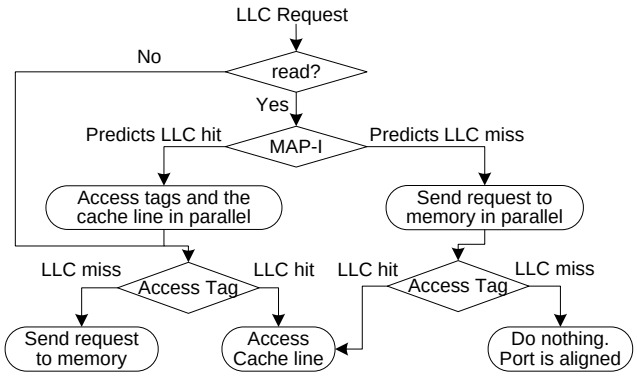


Fig. 6. Control flow of BlendCache

read hit, the tag and the cache line at location $L_j$ (highlighted by red arrow in Fig. 5) is accessed simultaneously. If the stored tag field matches with the requested memory block address (i.e., MAP-I correctly predicts an LLC hit), the data is forwarded to the core and the lower level caches. Otherwise, in case of a wrong LLC read hit prediction by MAP-I, the request is sent to the memory to service the LLC miss. The latency breakdown of BlendCache for correct and wrong LLC read hit prediction is shown in Fig. 4-(b) and Fig. 4-(c) respectively. For a correct LLC read prediction, BlendCache (cf. Fig. 4-b) reduces the LLC hit latency compared to the baseline LLC (cf. Fig. 4-a) due to simultaneous access of the tags and the data which are stored in the same RTM index.

If the MAP-I predicts an LLC read miss, BlendCache probes the RTM LLC (i.e., verification of an LLC miss requires a tag lookup in RTM) and memory in parallel (cf. Fig. 4-e) without accessing the cache line of location $L_j$. In this case, BlendCache quickly forwards the request to memory, thereby removing the high miss detection latency of RTM from the critical path. In case of a correct LLC read miss prediction by the MAP-I, the data returned from the memory is placed in RTM at location $L_j$ and the tag field is updated. At this point, the access port is likely to be aligned with location $L_j$ because the corresponding tag field was already accessed during the tag lookup phase. In case of a correct LLC read miss prediction, BlendCache reduces the LLC miss latency (Fig. 4-e) compared to the baseline LLC (Fig. 4-c). If the MAP-I makes a wrong LLC read miss prediction, only the data field of the relevant DBC row needs to be accessed. In this case, the data returned from the memory is ignored (cf. Fig. 4-f). We do not use MAP-I for non-critical prefetch and write requests. In this scenarios, tag and data are accessed serially as shown in Fig. 6.

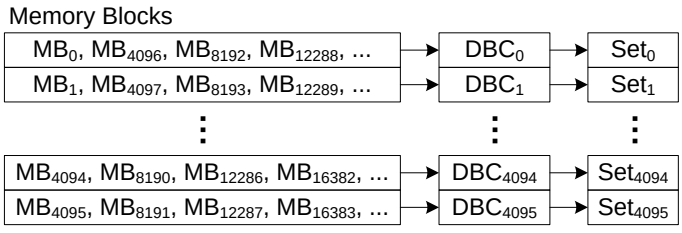


Fig. 7. Data mapping used in the baseline LLC

### C. Proposed Data Mapping

For a given LLC capacity and a single access port per track, increasing the number of domains per track (i.e., $K$) reduces the total number of access ports (or DBCs) which, in turn, leads to a lower leakage energy. However, the larger the value of $K$, the worse the overall LLC latency becomes due to the shift overhead (cf. Fig. 11 in Section IV-B). The baseline LLC incurs high shift overhead because the shifting pattern that results from the LRU policy is random in nature. The random shifting occurs because the main memory blocks that are mapped to the same DBC are spatially distant. This is depicted in Fig. 7 where spatially far main memory blocks

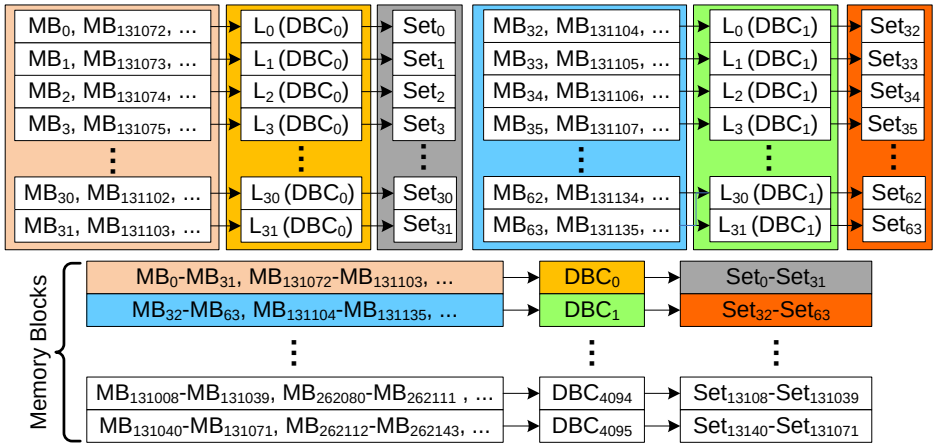


Fig. 8. BlendCache data mapping

(e.g., $MB_0$, $MB_{4096}$, $MB_{8196}$, ...) are mapped to the same DBC (e.g., $DBC_0$). Therefore, it is unlikely that subsequent accesses occur to the nearby locations within a particular DBC.

Since spatial adjacent memory blocks are likely to be accessed consecutively, our design maps them to adjacent locations within a DBC (i.e., adjacent domains within a track). This considerably reduces the RTM shifts. Fig. 8 shows how BlendCache maps main memory blocks to a DBC and a location within a DBC. For instance, main memory blocks $MB_0$, $MB_1$, $MB_2$, and $MB_3$ are mapped to locations $L_0$, $L_1$, $L_2$, and $L_3$ of $DBC_0$ respectively. Thus, the probability of temporally close accesses going to the nearby locations within a DBC is high.

### D. Overhead

The MAP-I [12] employed by BlendCache is used to keep track if the outcome of the an LLC access is a hit or a miss. The MAP-I in our implementation uses a table of 256 3-bit saturating *Memory Access Counters* (*MACs*). To get the desired *MAC*, the program counter (i.e., PC) that generated an LLC request is used to index the table using folded-xor hashing [13]. MAP-I does not require the PC to be stored in the LLC and requires the partial PC address to be propagated to MAP-I. The storage overhead of this MAP-I implementation accounts to 96 bytes (256 $\times$ 3 = 768 bits = 96 bytes) for a single-core system.

For a multi-core system, the MAC table counters are maintained on a per-core basis to avoid inter-core interference. For an *n*-core system, the total storage overhead of MAP-I is $n\times$ 96 bytes. This MAP-I is implemented in SRAM and requires around $0.0065\,\mathrm{mm}^2$ additional area, which is an overhead of (0.14%, 0.56%) for a (1,4) core system compared to the area of the BlendCache realized using multi-bit RTM cells ($4.46\,\mathrm{mm}^2$). The latency overhead to access MAP-I is 1-cycle with a dynamic energy of $3.2\,\mathrm{pJ}$/access and $1.5\,\mathrm{mW}$ leakage power. These overheads are estimated using CACTI6 [14] that includes the energy consumption of the SRAM and the interconnect (i.e., energy incurred to transfer the partial PC address bits to MAP-I). Since MAP-I is maintained on a per-core basis, we assume that it is located near private L2 cache to reduce interconnect energy. Note that MAP-I is accessed after a read miss in L2 cache.

## IV. Evaluation

This section presents the experimental setup used to perform iso-capacity comparison of BlendCache with the state-of-the-art. This section also analyzes the results of different experiments conducted on a simulation framework.

TABLE I
Configuration and Benchmark Details Used in the Experiments

| | |
|---|---|
| Core | 3.2 GHz, out-of-order, 4-issue |
| Private L1<br>Private L2<br>Shared RTM LLC | 32 KiB, 8-way associativity, 2-cycles latency<br>256 KiB, 8-way associativity, 4-cycles latency<br>8 MB, associativity and latency based on configuration |
| Main Memory (DRAM) | 2 channels, 16 KB row buffer, 64-bit channel width, tRAS-tRCD-tRP-tCAS-tWR = 35-11.25-11.25-11.25-11.25 (nsec) |
| SPEC2006 Benchmark (LLC APKI) | *sjeng* (1.17), *astar.t* (9.1), *leslie3d.r* (18), *leslie3d.t* (18.8), *milc*(23.8), *libquantum* (25.7), *soplex.r* (28.1),*omnetpp* (29.6), *astar.b* (49), *mcf* (66.9) |

### A. Experimental Methodology

The architectural simulations are carried out using a cycle-accurate multi-core simulator namely sim-zesto [15] employing the system configurations given in Table I. The workloads consists of 10 SPEC2006 benchmarks [16]. These applications vary in terms of LLC access per thousand instructions (LLC APKI) as listed in Table I. For instance, *mcf* (LLC APKI is 66.9) has high LLC access rate, while the least intensive is *sjeng* (1.17). Since this paper deals with the RTM LLC, the simulator has been modified to faithfully model shift operations. The modified simulator models separate tag/data array (to model the baseline LLC), a unified tag/data array with MAP-I predictor to model BlendCache. All the evaluated configurations employs SPEC2006 applications in single-core (cf. Table I) and multi-core mode (cf. Table V and Section IV-G).

TABLE II
DEVICE AND ELECTRICAL PARAMETERS OF THE RTM CELL AND THE MOS TRANSISTOR BASED ON THE MODEL IN [5] FOR 45 nm . F IS THE TECHNOLOGY FEATURE SIZE

| Parameter | Value |
|---|---|
| Width of the racetrack | 1F |
| Gap distance between two racetracks | 2F |
| Length of the domain in a racetrack | 2F |
| Thickness of the racetrack | 6 nm |
| Gate width of the MOS transistor | 9F |
| Length of the MOS transistor | 3F |
| Gap distance between two MOS transistors | 1F |
| PMA racetrack nanowire resistivity | $4.8 \times 10^{-7} \Omega m$ |
| Critical current density for shift | $6.2 \times 10^{7} A/m^2$ |
| Resistance-area product | $10 \Omega \mu m^2$ |
| Cell turn-on resistance | $3000\Omega$ |
| Cell turn-off resistance | $6000\Omega$ |
| Current sense amplifier read voltage | 0.25 V |
| Minimum sense voltage | 0.025 V |
| Read power per bitline | 20 µW |
| Cell set/reset current | 80 µA |
| Cell set/reset pulse | 0.5 ns |

Since SPEC2006 applications are single-threaded, one instance of each application is run per core. For each core, we employ a reservation station with 32 entries, load queue with 32 entries, store queue with 24 entries and reorder buffer with 80 entries. Each core can fetch, decode and commit maximum of four x86 instructions in a single cycle. We fast-forward each application to region of interest using Simpoint tool [17] and warm up all the caches for 500 million instructions. We simulate one billion instructions in a single-core and a total of five billion instructions in a multi-core mode. In the multi-core mode, shorter workloads are relaunched whey they finish early by completing their instructions to keep the overall system loaded. The statistics gathered from sim-zesto including total accesses, total shifts and overall runtime are used to estimate the overall LLC energy consumption. We employ a per core stream prefetcher similar to [18] for all evaluated configurations. We use 8, 8, and 32 miss status handling registers (MSHRs) for L1, L2, and LLC respectively. Therefore, the maximum number of in-flight misses that can be handled by L1, L2, and LLC are 8, 8, and 32 respectively. We employed non-inclusive policy for the multi-level caches because it performs better compared to inclusive policy [19]. We make the following assumptions for all evaluated configurations including BlendCache:

1) The evaluation is performed for an 8 MB RTM LLC.
2) We assume 4 banks (i.e., $B$ = 4), 16 subarrays per bank (i.e., $S$ = 16), and a 64-byte cache line.
3) We vary the number of domains within a track (i.e., $K$ is the track length) and the number of DBCs per subarray (i.e., $D$) such that the LLC capacity remains 8 MB (i.e., $K \times D = 2048$). We evaluate three RTM organizations with $K$ = 8 (i.e., $D$ = 256), $K$ = 16 (i.e., $D$ = 128), and $K$ = 32 (i.e., $D$ = 64).

We compare BlendCache (i.e., BLEND-K) with the baseline LLC (i.e., BASE-K). We have estimated the energy and area numbers of the SRAM tag array in the baseline LLC using CACTI6 [14] assuming a uniform cache access model with 4 banks while the optimization target is set to leakage. The energy and area numbers of BlendCache and the RTM data array in the baseline LLC are estimated using a modified version of destiny [20] with read latency optimization target. The device and electrical parameters of the RTM cell and the access transistor (i.e., MOS) are provided in Table II based on the model described in [5].

The area, energy and the latency numbers of different evaluated configurations are are listed in Table III. A higher $K$ requires less number of access ports (or DBCs). As shown in Table III, the RTM design with $K$ = 8 is optimized for shift energy because it employs less number of domains per track. In contrast, the RTM design with $K$ = 32 is optimized for leakage power and area as it employs less number of leaky and area-hungry access ports. We also evaluate an idealistic RTM configuration (referred to as RTMIdeal) which is optimized both for latency (i.e., single-cycle tag access latency and zero shift penalty) and LLC miss rate (i.e., 32-way associativity). The baseline BASE-K has an SRAM tag access latency of 5 cycles (cf. Table III) with $K$-way associativity while the direct-mapped BLEND-K requires a single-cycle latency (i.e., to access MAP-I) before accessing the RTM array.

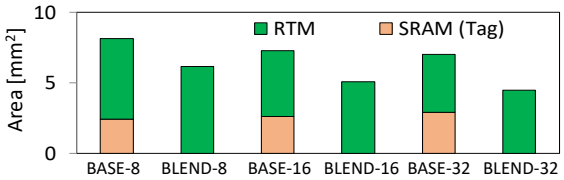


Fig. 9. LLC area results

### B. Result Overview

Before delving into details, we first discuss the overall impact of different LLC configurations on various performance metrics when $K$ is varied from 8 to 32. Fig. 9 shows the area benefits of BlendCache compared to the baseline LLC. As shown, BLEND-8, BLEND-16, and BLEND-32 improves the overall area by 24%, 37%, and 45% compared to BASE-8. This area improvement is due to the fact that BlendCache stores the tags along with the data in the denser RTM array, while the baseline LLC stores them in a separate SRAM array. The area of the SRAM tag array accounts for a significant portion of the total area in the baseline LLC because the area efficiency of SRAM is significantly lower than RTM. Since the cell density of RTM increases with $K$, so does the area efficiency of BlendCache.

The RTM LLC energy consumption highly depends on the total shift cost, which impacts the shift energy, and the total number of access ports/DBCs, which impacts the leakage energy. Fig. 10 shows the impact of the track length on the energy consumption of different RTM configurations. The energy results shows that BLEND-8, BLEND-16, and BLEND-32 improves the overall energy consumption by 28%, 44%, and 48% compared to BASE-8. We make the following observations from the energy results. First, the energy reduction of BlendCache is due to energy saving in both the leakage

TABLE III
ENERGY AND AREA OF VARIOUS RTM CONFIGURATIONS

| RTM Array (8 MB LLC, 45 nm, 512 bytes / cache line) | | | | | | |
|---|---|---|---|---|---|---|
| Configuration | BLEND-8 | BLEND-16 | BLEND-32 | BASE-8 | BASE-16 | BASE-32 |
| Leakage power [mW] | 241 | 181 | 155 | 224 | 167 | 142 |
| Read latency [ns/cycles] | 2.17/7 | 2.10/7 | 2.03/7 | 2.01/7 | 1.95/7 | 1.88/7 |
| Write latency [ns/cycles] | 3.70/12 | 3.59/12 | 3.50/12 | 3.62/12 | 3.51/12 | 3.43/11 |
| Read energy [pJ] | 290 | 283 | 274 | 271 | 263 | 254 |
| Write energy [pJ] | 437 | 421 | 405 | 412 | 397 | 380 |
| Shift energy [pJ] | 178 | 189 | 206 | 164 | 173 | 187 |
| Area [$mm^2$] | 5.71 | 4.68 | 4.11 | 6.15 | 5.07 | 4.46 |
| Tag Array (SRAM for the baseline LLC) | | | | | | |
| Configuration | BLEND-8 | BLEND-16 | BLEND-32 | BASE-8 | BASE-16 | BASE-32 |
| Leakage power [mW] | NA | NA | NA | 120.5 | 126.3 | 131.6 |
| Tag energy [pJ] | NA | NA | NA | 46.76 | 49.51 | 51.63 |
| Tag latency [ns/cycles] | NA | NA | NA | 1.31/5 | 1.42/5 | 1.50/5 |
| Area [$mm^2$] | NA | NA | NA | 2.40 | 2.58 | 2.89 |

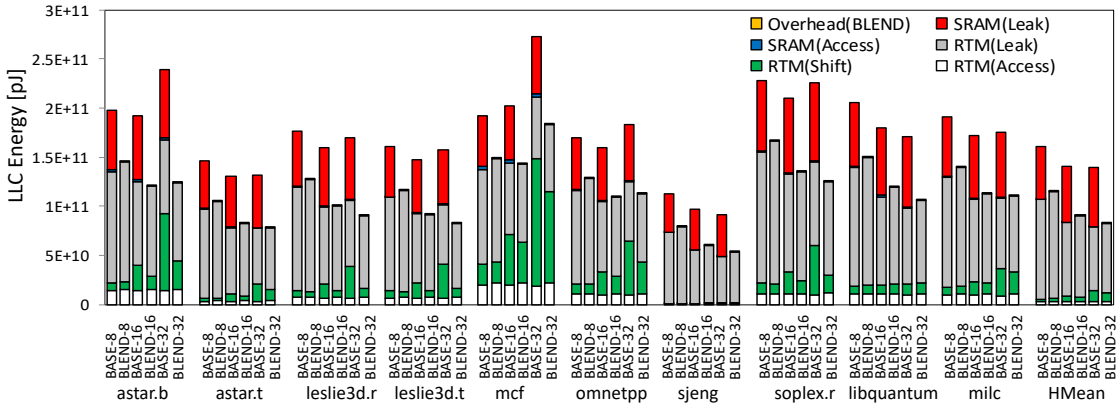


Fig. 10. LLC Energy breakdown

and the shift energy. The improvement in the leakage energy is provided by storing the tags in the less leaky RTM cells (cf. Section III-A and Fig. 5) while the shift energy gain is delivered by reduction in the total number of shifts (cf. Fig. 11 and Section IV-C). Second, the leakage energy of the SRAM tag array accounts for a significant portion of the total energy in the baseline LLC. Third, the energy saving for BlendCache is more pronounced for leakage-optimized RTM configurations (e.g., BLEND-32) as they have less access ports. Employing BLEND-32, the increase in the shift energy (via a higher shift cost) compared to BLEND-8 is offset by significant reduction in the leakage energy. Fourth, the extra energy overhead of the MAP-I used by BlendCache is less than 1% for all variants of $K$.

The instruction per cycle (IPC) results can be observed in Fig. 12 which shows that the harmonic mean IPC degradation of BlendCache compared to RTMIdeal is 1.6%, 1.9% and 2.2% for a track of length 8, 16, and 32, respectively. The IPC results indicate that the worst case IPC degradation of BlendCache is 5.7% (for *omnetpp* employing BLEND-32). The performance of the baseline LLC compared to RTMIdeal is 0.9%, 1.2%, and 1.8% for $K$ = 8, 16, and 32, respectively. Thus, the baseline $K$-way LLC slightly outperforms the direct-mapped BlendCache due to lower LLC MPKI (cf. Fig. 14). However, BlendCache provides a significant improvement in the LLC area (Fig. 9) and the LLC energy consumption (Fig. 10).

### C. Total Shift Cost

The RTM shift cost primarily depends on how well the spatial locality of applications is exploited. BlendCache exploits the spatial locality by mapping adjacent memory blocks to consecutive locations in RTM (cf. Fig. 8). This ensures that the shift distance in subsequent memory accesses is minimized. On the contrary, the baseline LLC maps spatially far memory blocks to the same cache set (cf. Fig. 7 and Section III-C). Due to this reason, the probability of temporally close accesses going to nearby locations is very high in BlendCache compared to the baseline LLC. In addition, the random shifting nature of the LRU policy used by the baseline LLC further exacerbates the total shift cost. As a result of mostly sequential access pattern in BlendCache, the overall shift cost is reduced compared to the baseline LLC. As shown in Fig. 11, BlendCache reduces the total shift cost by 6.2%, 19.3%, and 31.5% compared to the baseline LLC for $K$ = 8, 16, and 32, respectively. The shift reduction of BlendCache is more pronounced for longer track length (i.e., higher $K$) compared to the shorter one.

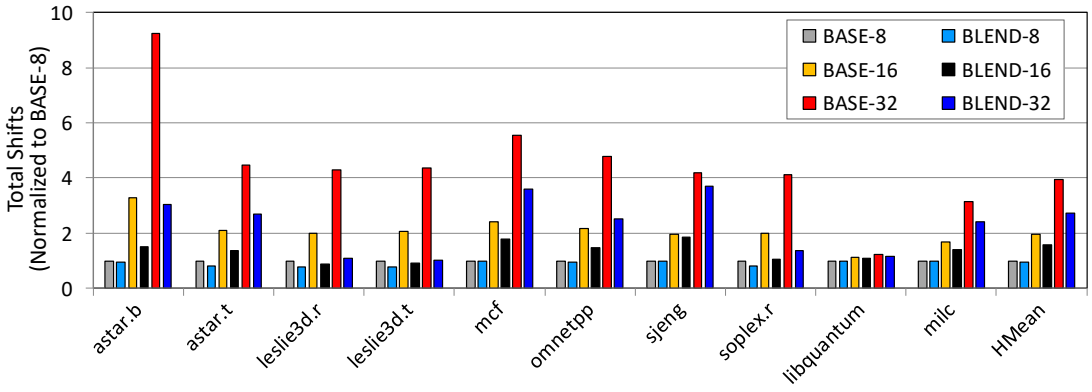


Fig. 11. Total shift cost normalized to BASE-8

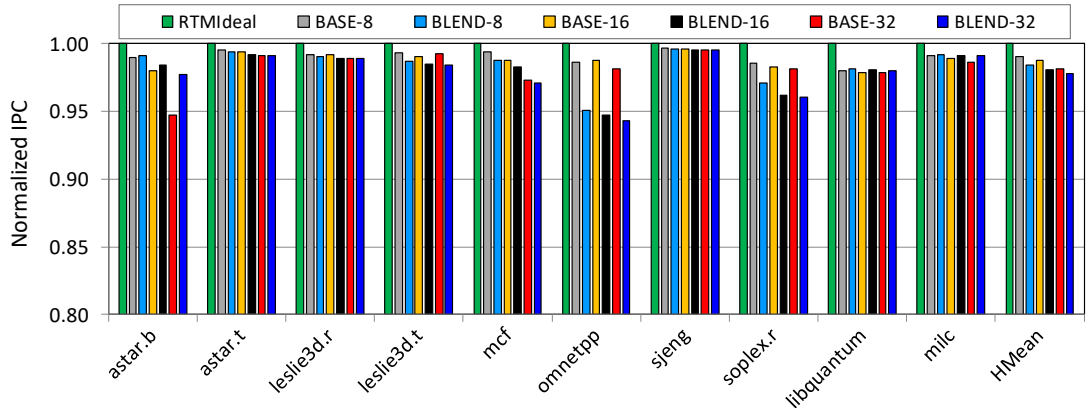


Fig. 12. Instruction per cycle (IPC) results

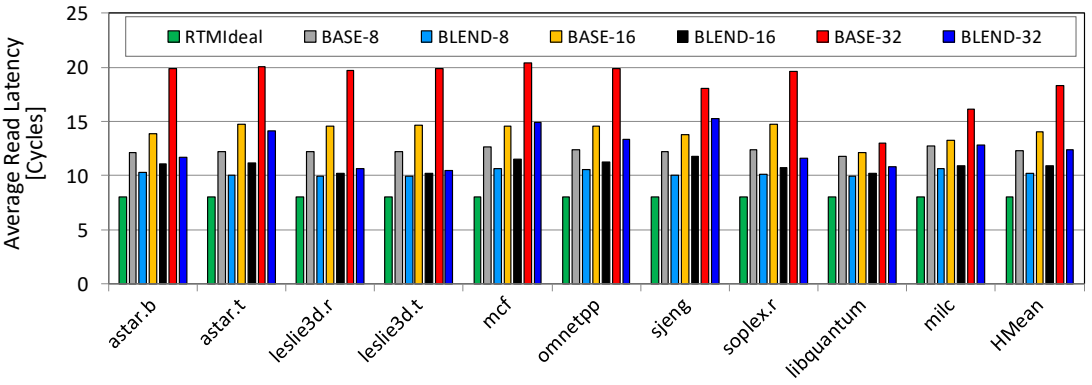


Fig. 13. Average LLC read access latency

### *D. LLC Read Access Latency*

As discussed in Section III and shown in Fig. 4, a correct prediction made by MAP-I reduces LLC hit/miss latency compared to the baseline LLC. On average, the prediction accuracy of MAP-I is around 94%. BlendCache also reduces the shift cost per read access via exploiting spatial locality. This results in improved average read latency compared to the baseline LLC as depicted in Fig. 13. As shown, the read latency degradation of the baseline LLC with respect to RTMIdeal is 53%, 75%, and 129% for an 8-bit, 16-bit, and 32-bit track respectively. BlendCache cuts this latency degradation by 27%, 36%, and 54% for $K$ = 8, 16, and 32 respectively.

### *E. Application Characteristics*

The performance of a particular application depends on its sensitivity to read access latency and miss rate. Based on these metrics, the applications can be categorized into two types namely *latency-critical* and *miss-critical*. The performance of latency-critical applications (e.g., *astar.b*) is highly influenced by read access latency and less affected by miss rate. These application have high APKI (cf. Table I) and low MPKI (cf. Fig 14). For instance, the APKI of the latency-critical *astar.b* is 49 its MPKI is less than 2. Since BlendCache is optimized for read access latency, it outperforms the baseline LLC for latency-critical *astar.b*. In contrast, performance of

miss-critical applications (e.g., *omnetpp* and *soplex.r*) is highly sensitive to miss rate and less sensitive to read access latency. The MPKI of these applications is negatively impacted by reducing associativity. Since the direct-mapped BlendCache increases the MPKI of miss-critical applications compared to $K$-way associative baseline LLC (cf. Fig 14), it performs worse compared to the baseline LLC for these applications.

The energy of a particular application depends on its LLC intensity. For instance, for the applications with low LLC intensity (e.g., sjeng requires fewer LLC accesses with LLC APKI of 1.17), the contribution of leakage energy to the overall energy is large while the shift energy contribution is small. On the contrary, the contribution of RTM shift energy is more pronounced for applications with high LLC intensity (e.g., *mcf*, *astar.b*, and *omnetpp*). For these applications, BLEND-16 outperforms BLEND-32 in terms of overall energy consumption because the leakage energy advantage of BLEND-32 is negatively offset by an increase in the shift energy. For the remaining applications with relatively low LLC intensity, BLEND-32 provides more energy saving compared to BLEND-8 and BLEND-16 because the contribution of the shift energy in these application is less prominent compared to the leakage energy.

TABLE IV
PARAMETERS FOR THE TAG ARRAY REALIZED USING SINGLE-BIT RTM CELLS IN TAPECACHE

| Configuration | TAPE-8 | TAPE-16 | TAPE-32 | TAPE-DM |
|---|---|---|---|---|
| Tag storage [KiB] | 528 | 560 | 592 | 432 |
| Leakage [mW] | 51.5 | 54.6 | 57.3 | 43.8 |
| Tag energy [pJ] | 33.4 | 35.6 | 36.9 | 29.2 |
| Tag latency [ns] | 1.07 | 1.14 | 1.21 | 0.98 |
| Tag latency [cycles] | 4 | 4 | 4 | 4 |
| Area [$mm^2$] | 0.95 | 1.04 | 1.15 | 0.79 |

### F. Comparison with TapeCache

TapeCache [3] stores the data in multi-bit RTM cells and addresses the leakage problem by storing the tags in a separate RTM array comprised of single-bit RTM cells incurring zero shift cost [3]. The single-bit RTM tag array has reduced leakage power compared to an SRAM tag array (cf. Table III and Table IV). Fig. 15 and Fig. 16 compares the energy consumption and the performance of BlendCache with two variants of TapeCache (i.e., TAPE-K with K-way associativity and TAPE-DM with direct-mapped design). As shown, TAPE-K achieves noticeable energy reduction compared to the baseline LLC which is mainly achieved via reduction in the leakage energy. This is because the leakage of the SRAM tag array employed in the baseline LLC is significantly higher compared to the single-bit RTM tag array employed in TAPE-K. To further mitigate the leakage energy problem, BlendCache stores the tags along with the data in the leakage optimized RTM cells. As a result, BlendCache reduces the overall energy consumption by 10.8%, 16.2%, and 20.8% compared to TAPE-K for a track of length 8, 16, and 32 bits respectively. In addition, the area improvement compared to TAPE-K is 7.6%, 11.4%, and 15.2%. The performance results in Fig. 16 shows that BlendCache provides comparable performance to TapeCache due to the latency optimizations discussed in Section III.

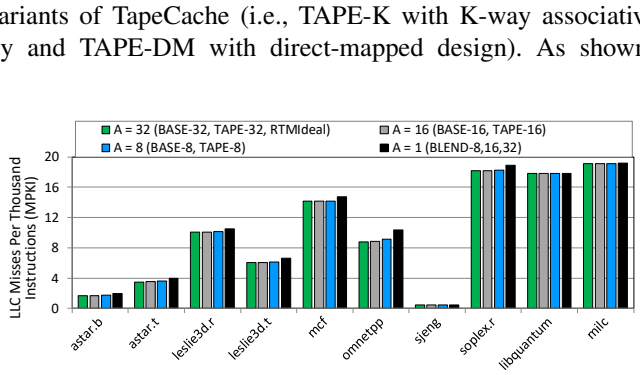


Fig. 14. LLC misses per thousand instructions (LLC MPKI) for various LLC associativity

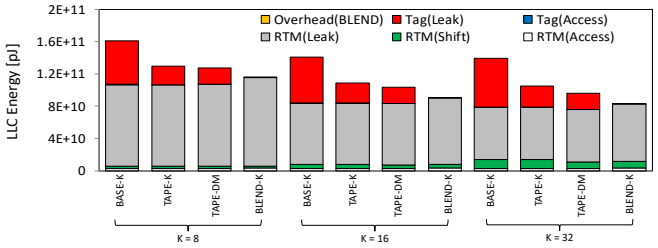


Fig. 15. Energy comparison with TapeCache [3]. The energy is averaged over all applications

Similar to BlendCache, TAPE-DM employs the same data mapping in Fig. 8 and exploits the spatial locality of applications. As a result, TAPE-DM reduces the shift energy compared to TAPE-K and the baseline LLC. Since the tag storage requirement for direct-mapped TAPE-DM is smaller compared to K-way associative TAPE-K, TAPE-DM reduces the leakage energy dissipated in the tag array. As shown in Fig. 15, BlendCache reduces the energy consumption by 9.2% ($K$ = 8), 12.4% ($K$ = 16), and 13.5% ($K$ = 32) compared to TAPE-DM. Another advantage of BlendCache over TAPE-DM is that it serves LLC hits and misses faster for the majority of the LLC requests (i.e., when MAP-I makes correct LLC hit/miss prediction), thereby slightly outperforming TAPE-DM for all track lengths.

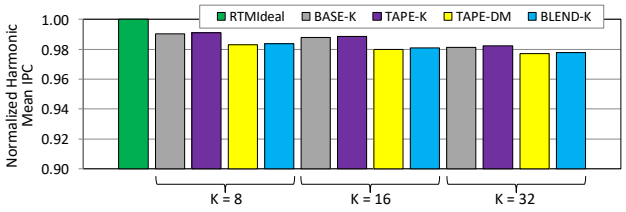


Fig. 16. Performance comparison with TapeCache [3]. The normalized performance is averaged over all applications

### G. Multi-core Results

For the evaluation of multi-core systems, we constiute five benchmark mixes by combining four different applications which are listed in Table V. The harmonic mean instruction per cycle (HMIPC) results in Fig. 17 shows that the performance

TABLE V
SPEC2006 BENCHMARK MIXES

| Name | Benchmark mix |
|---|---|
| Mix1 | *leslie3d.r*, *omnetpp*, *mcf*, *libquantum* |
| Mix2 | *astar.b*, *milc*, *soplex.r*, *astar.t* |
| Mix3 | *astar.t*, *leslie3d.t*, *sjeng*, *libquantum* |
| Mix4 | *astar.b*, *leslie3d.r*, *omnetpp*, *leslie3d.t* |
| Mix5 | *mcf*, *milc*, *sjeng*, *soplex.r* |

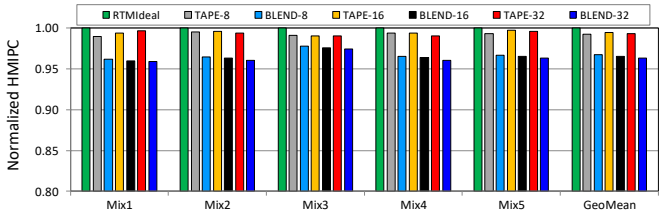


Fig. 17. Normalized Harmonic Mean Instruction per cycle (HMIPC) results for benchmark mixes listed in Table V

degradation of BlendCache compared to TapeCache is 2.5%, 2.9% and 3% for a track of length 8, 16, and 32, respectively. Compared to single-core mode, the performance degradation of BlendCache compared to TapeCache is higher in multi-core mode which is caused by its direct-mapped nature. However, this performance degradation is largely compensated by significant energy saving compared to TapeCache for longer track length. As shown in Fig. 18, BlendCache reduces the energy consumption by 9.7%, 22.5%, and 35.9% compared to TapeCache for $K$ = 8, 16, and 32, respectively. Employing TapeCache, the shift energy dominates the total energy consumption which is caused by higher shift cost. Since increasing the number of cores put more pressure on the LLC, the shift energy in TapeCache is more prominent in multi-core mode (cf. Fig. 18) compared to single-core mode (Fig. 10). BlendCache reduces the number of shift operations

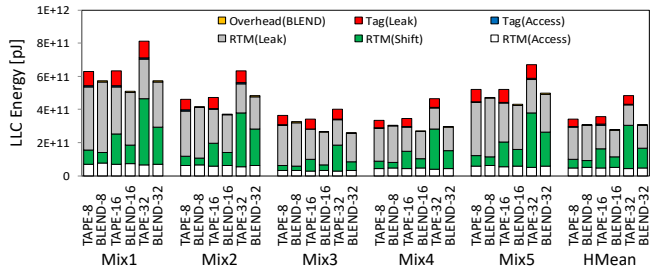


Fig. 18. LLC Energy breakdown for benchmark mixes listed in Table V

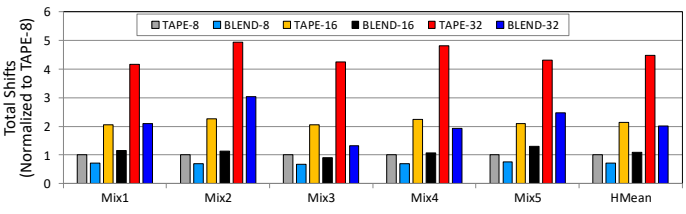


Fig. 19. Total shift cost normalized to TAPE-8 for benchmark mixes listed in Table V

compared to TapeCache (cf. Fig. 19), in particular, for longer track length which reduces the shift energy. Compared to TapeCache, BlendCache also reduces the leakage energy by storing the tags in the less leaky RTM cells.

## V. RELATED WORK

RTM has been employed at different levels in the memory hierarchy including register file [21]–[23], scratchpads [24]–[26], caches [1]–[6], [27], [28], network-on-chip [29], off-chip memory [30], and SSD [31]. Many architectural techniques have been employed for shift reduction that include preshifting [22], [32], access port management [1], [3], [6] and data migration [3], [6]. In addition, optimizations at cell-level [6], circuit-level [27], layout-level [5], [28], and cross-level [7] have been applied to provide performance, energy and area improvements.

The architectural solutions for shift cost reduction in RTM caches primarily reduces the shift component of energy which makes them suitable for the smaller caches [1], [3], [6], [22], [32]. Due to the higher cache access intensity in the smaller caches, their energy consumption is primarily dominated by the shift energy and least impacted by the leakage energy. The access intensity of the larger LLC is less compared to the smaller private caches (i.e., L1 and L2) because the private caches filter out majority of the cache traffic from the LLC. Therefore, the energy consumption of larger LLC is highly impacted by the leakage energy. To the best of our knowledge, this is the first work which enables efficient usage of the leakage optimized RTM cells for the tag storage to reduce the overall LLC energy consumption. It is worth to mention that the aforementioned architectural approaches for shift reduction are orthogonal to our work and can be applied on top of BlendCache.

TapeCache analyzed the impact of shift operations and proposed an interleaved organization (also adopted in BlendCache) to store the bits of the cache lines [1], [3]. In addition, to mitigate the negative impact of shifting overhead on performance, they propose access port management policy based on next cache line prediction by preshifting the port position to a cache line which is likely to be accessed in future. Similarly, other preshifting methods have been proposed to mitigate the impact of shift penalty on the performance [22], [32]. However, preshifting increases the overall energy consumption for a wrong next cache line prediction. Also, the performance of applications is less impacted by latency at LLC level which makes the preshifting optimizations less attractive at LLC level.

Previous studies have employed direct-mapped cache design on top of SRAM [33] and DRAM caches [12] by showing that a direct mapped cache design has the capability to outperform set associative designs in terms of performance. While the characteristics and constraints of RTM are quite different from SRAM and DRAM, this works make a case for direct-mapped RTM LLC and showed that an intelligent direct-mapped design outperforms complex set associative and naive direct-mapped designs in terms of energy consumption.

## VI. Conclusions

The leakage energy problem of an RTM based LLC can be mitigated by storing the tags in the multi-bit RTM cells. However, these cells exhibit a high probability of long shift cost which makes it difficult to use them for the latency critical tag array. To overcome this limitation, we develop BlendCache that efficiently enables the tag storage in the multi-bit RTM cells, providing substantial reduction in the energy consumption. The direct-mapped nature of BlendCache does not require a tag lookup to determine the target location in RTM which reduces the LLC hit latency. The negative impact of high tag lookup latency to service an LLC read miss in BlendCache is mitigated by sending majority of these requests to memory before performing the costly tag check in RTM. In addition, the overall shift cost in RTM is minimized via efficient data mapping that exploits programs spatial locality by assigning consecutive blocks from main memory to adjacent locations in RTM. These latency optimizations make the tag store in the leakage optimized multi-bit RTM cells practical, thereby achieving high energy efficiency.

## Acknowledgments

This work was partially funded by the German Research Council (DFG) through the DART-HMS project (project number 437232907) and the TraceSymm project (project number 366764507).

## References

[1] R. Venkatesan, V. Kozhikkottu, C. Augustine, A. Raychowdhury, K. Roy, and A. Raghunathan, "TapeCache: A High Density, Energy Efficient Cache Based on Domain Wall Memory," in *Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design*, ser. ISLPED '12, 2012, pp. 185–190.

[2] H. Xu, Y. Alkabani, R. Melhem, and A. K. Jones, "Fusedcache: A naturally inclusive, racetrack memory, dual-level private cache," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 2, no. 2, pp. 69–82, April 2016.

[3] R. Venkatesan, V. J. Kozhikkottu, M. Sharad, C. Augustine, A. Raychowdhury, K. Roy, and A. Raghunathan, "Cache Design with Domain Wall Memory," *IEEE Transactions on Computers*, vol. 65, no. 4, pp. 1010–1024, 2016.

[4] H. Xu, Y. Li, R. Melhem, and A. K. Jones, "Multilane Racetrack Caches: Improving Efficiency Rhrough Compression and Independent Shifting," in *The 20th Asia and South Pacific Design Automation Conference*, Jan 2015, pp. 417–422.

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

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

[7] G. Sun, c Zhang, H. Li, Y. Zhang, W. Zhang, Y. Gu, Y. Sun, J. Klein, D. Ravelosona, Y. Liu, W. Zhao, and H. Yang, "From Device to System: Cross-Layer Design Exploration of Racetrack Memory," in *DATE*. ACM, 2015, pp. 1018–1023.

[8] S. Parkin and S.-H. Yang, "Memory on the Racetrack," vol. 10, pp. 195–198, 03 2015.

[9] R. Bläsing, A. A. Khan, P. C. Filippou, C. Garg, F. Hameed, J. Castrillon, and S. S. P. Parkin, "Magnetic racetrack memory: From physics to the cusp of applications within a decade," *Proceedings of the IEEE*, pp. 1–19, Mar. 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9045991

[10] Y. Chee, R. Gabrys, A. Vardy, V. Vu, and E. Yaakobi, "Reconstruction from deletions in racetrack memories," in *2018 IEEE Information Theory Workshop (ITW)*, 2018, pp. 1–5.

[11] Y. Chee, H. Kiah, A. Vardy, V. Vu, and E. Yaakobi, "Coding for racetrack memories," *IEEE Transactions on Information Theory*, vol. 64, no. 11, pp. 7094–7112, 2018.

[12] M. Qureshi and G. Loh, "Fundamental Latency Trade-offs in Architecting DRAM Caches," in *45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2012, pp. 235–246.

[13] A. Seznec and P. Michaud, "A Case for (Partially) TAgged GEometric History Length Branch Prediction," *Journal of Instruction Level Parallelism*, vol. 8, 2006.

[14] N. Muralimanohart and N. Balasubramonian, R.and Jouppi, "Optimizing NUCA Organizations and Wiring Alternatives for Large Caches with CACTI 6.0," in *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, December 2007, pp. 3–14.

[15] G. Loh, S. Subramaniam, and Y. Xie, "Zesto: A Cycle-Level Simulator for Highly Detailed Microarchitecture Exploration," in *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2009.

[16] "Standard Performance Evaluation Corporation," http://www.spec.org, 2021, [Online; accessed 25-May-2021].

[17] G. Hamerly, E. Perelman, J. Lau, and B. Calder, "Simpoint 3.0: Faster and More Flexible Program Analysis," *Journal of Instruction Level Parallelism*, vol. 7, 2005.

[18] R. X. Arroyo, R. J. Harrington, S. P. Hartman, and T. Nguyen, "IBM POWER7 Systems," *IBM Journal of Research and Development*, vol. 55, no. 3, pp. 2:1 – 2:13, 2011.

[19] A. Jaleel, E. Borch, M. Bhandaru, S. Steely, and J. Emer, "Achieving Non-Inclusive Cache Performance with Inclusive Caches," in *Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, December 2010, pp. 151–162.

[20] S. Mittal, R. Wang, and J. Vetter, "DESTINY: A Comprehensive Tool with 3D and Multi-Level Cell Memory Modeling Capability," *Journal of Low Power Electronics and Applications*, vol. 7, no. 3, 2017.

[21] M. Mao, W. Wen, Y. Zhang, Y. Chen, and H. Li, "Exploration of GPGPU Register File Architecture Using Domain-wall-shift-write Based Racetrack Memory," in *Proceedings of the 51st Annual Design Automation Conference on Design Automation Conference*, ser. DAC '14, 2014, pp. 196:1–196:6.

[22] E. Atoofian, "Reducing Shift Penalty in Domain Wall Memory Through Register Locality," in *Proceedings of the 2015 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, ser. CASES '15, 2015, pp. 177–186.

[23] M. Moeng, H. Xu, R. Melhem, and A. Jones, "ContextPreRF: Enhancing the Performance and Energy of GPUs With Nonuniform Register Access," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, pp. 343–347, 2016.

[24] A. A. Khan, N. A. Rink, F. Hameed, and J. Castrillon, "Optimizing Tensor Contractions for Embedded Devices with Racetrack Memory Scratch-pads," in *Proceedings of the 20th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems*, ser. LCTES 2019, 2019, pp. 5–18.

[25] H. Mao, C. Zhang, G. Sun, and J. Shu, "Exploring Data Placement in Racetrack Memory Based Scratchpad Memory," in *2015 IEEE Non-Volatile Memory System and Applications Symposium (NVMSA)*, Aug 2015, pp. 1–5.

[26] A. A. Khan, F. Hameed, R. Bläsing, S. S. P. Parkin, and J. Castrillon, "ShiftsReduce: Minimizing Shifts in Racetrack Memory 4.0," *ACM Transactions on Architecture and Code Optimization*, vol. 16, no. 4, Dec. 2019. [Online]. Available: https://doi.org/10.1145/3372489

[27] S. Motaman, A. Iyengar, and S. Ghosh, "Synergistic Circuit and System Design for Energy-efficient and Robust Domain Wall Caches," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, ser. ISLPED '14, 2014, pp. 195–200.

[28] Z. Sun, X. Bi, A. K. Jones, and H. Li, "Design Exploration of Racetrack Lower-Level Caches," in *2014 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2014, pp. 263–266.

[29] D. Kline, H. Xu, R. Melhem, and A. K. Jones, "Domain-wall memory buffer for low-energy nocs," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015, pp. 1–6.

[30] Q. Hu, G. Sun, J. Shu, and C. Zhang, "Exploring Main Memory Design Based on Racetrack Memory Technology," in *2016 International Great Lakes Symposium on VLSI (GLSVLSI)*, May 2016, pp. 397–402.

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

[32] A. Colaso, P. Prieto, P. Abad, J. A. Gregorio, and V. Puente, "Architecting Racetrack Memory Preshift through Pattern-Based Prediction

Mechanisms," in *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2019, pp. 273–282.

[33] M. Hill, "A Case for Direct-mapped Caches," *IEEE Computer*, vol. 21, no. 12, pp. 25–40, December 1988.

**Fazal Hameed** Fazal Hameed received his Ph.D. (Dr.-Ing.) degree in computer science from the Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany, in 2015. He joined the chair for Compiler Construction at the TU Dresden (Dresden, Germany) as Post-doctoral researcher in March 2016. Before, he worked on a similar position at the Chair of Dependable and Nano Computing (CDNC) Karlsruhe Institute of Technology (KIT), Germany. He is currently affiliated with Institute of Space Technology, Islamabad, Pakistan. He mainly works in the architecture group with a focus on memories. Mr. Hameed was a recipient of the CODES+ISSS 2013 Best Paper Nomination for his work on DRAM cache management in multicore systems. He has served as an External Reviewer for major conferences in embedded systems and computer architecture.

**Jeronimo Castrillon** Jeronimo Castrillon is a professor in the Department of Computer Science at the TU Dresden, where he is also affiliated with the Center for Advancing Electronics Dresden (CfAED). He is the head of the Chair for Compiler Construction, with research focus on methodologies, languages, tools and algorithms for programming complex computing systems. He received the Electronics Engineering degree from the Pontificia Bolivariana University in Colombia in 2004, the master degree from the ALaRI Institute in Switzerland in 2006 and the Ph.D. degree (Dr.-Ing.) with honors from the RWTH Aachen University in Germany in 2013. Prof. Castrillon served in the executive committee of the ACM "Future of Computing Academy" from 2017 to 2019.