A Novel Hybrid DRAM/STT-RAM Last-Level-Cache Architecture for Performance, Energy and Endurance Enhancement

Fazal Hameed, Jeronimo Castrillon

***Abstract*—High capacity L4 architectures as Last-Level-Cache (LLC) have been recently introduced between L3-SRAM and off-chip memory. These LLC architectures have either employed DRAM or Spin-Transfer-Torque (STT-RAM) memory technologies. It is a known fact that DRAM LLCs feature a higher energy consumption while STT-RAM LLCs feature a lower write endurance compared to their counterparts. This paper proposes an efficient hybrid DRAM/STT-RAM LLC architecture that exploits the best characteristics offered by the individual memory technologies while mitigating their drawbacks. More precisely, we introduce a novel mechanism for the storage and management of the hybrid LLC tags, and a proactive L3-SRAM writeback policy that combines multiple dirty blocks that are mapped to the same LLC row. Our hybrid architecture reduces LLC interference by having less writeback accesses and row fetches. The endurance is improved by reducing the number of STT-RAM block writes. We show that our LLC architecture reduces the total number of STT-RAM block writes by 78% and improves the average performance by 13% compared to a recently proposed STT-RAM LLC. Compared to the state-of-the-art DRAM LLC, we report an average energy and performance improvement of 24% and 17.1% respectively.**

***Index Terms*—Architecture, cache, memory, memory hierarchy.**

I. Introduction

The increasing processor-memory speed gap has made memory accesses extremely expensive. In addition, the increase in the memory and bandwidth requirements of emerging applications [1] has further magnified the impact of the gap. Therefore, the miss rate of traditional SRAM-based **L**ast-**L**evel-**C**ache (LLC) has worsened due to its limited capacity. One way to mitigate this problem is to reduce the number of off-chip accesses by employing an additional level of high capacity L4 cache ($\geq$ 128MB) on top of L1/L2/L3 SRAM caches. Therefore, die-stacked memory technologies have been employed as LLC [2]–[7] since they provide a natural way to integrate high capacity memory on top of the processor die.

Fig. 1 depicts a typical multi-core cache hierarchy used in [3], [4], [8]. A larger LLC (DRAM or STT-RAM) is composed of many banks where each bank is provided with a **R**ow **B**uffer (RB). When an access is made to an LLC bank, one row of the bank is fetched into the bank's RB. The data in the RB can be accessed at much lower latency and lower energy than accessing it from the DRAM bank. State-of-the-art LLC use a large RB size (i.e. 2KB or 4KB) per LLC bank. This *large RB organization* incurs high energy consumption via buffering large number of unnecessary data. To reduce energy consumption, state-of-the-art employs multiple small subarrays (each provided with a RB) instead of a single large RB per bank [9], [10]. This *small RB organization* has shown improved performance benefits compared to the large RB organization in the context of off-chip memory due to an improved subarray-level-parallelism. However, we found that employing small RB organization [9] using existing LLC tag designs [4]–[6], [8] incur performance degradation due to a high tag lookup latency. To address this problem, we rethink the LLC tag design to make it viable for small RB organization.

Recent works have focused on relatively small hybrid SRAM/STT-RAM architectures for on-chip LLCs [11]–[16], e.g., 32 KB to 8 MB. To the best of our knowledge, hybrid DRAM/STT-RAM architectures have not been explored for larger on-chip LLCs, e.g. 256 MB. For large hybrid DRAM/STT-RAM LLCs, tag management is key to strike a good trade-off between storage, latency, endurance and energy consumption. For instance, a 256 MB cache would require 24 MB of tag storage assuming conventional 64-byte block size. One design option is to store the tags in a dedicated SRAM memory to lower access latency. However, this design is impractical due to large area and energy (especially leakage) requirements [5]. Storing the tags in STT-RAM, on the other hand, would exacerbate its write endurance [17]–[20] because tags are written more frequently than data. An obvious solution would be to store the LLC tags in a DRAM memory, not only avoiding the area and energy penalty of storing tags in SRAM but also addressing the STT-RAM write-endurance issue. However, a naive LLC tag design in DRAM may waste memory space due to internal fragmentation which is caused by leaving unused space in the LLC row.

In addition, existing LLC architectures suffer from performance degradations due to row buffer interference (see Section II-A) which is caused by increased number of writebacks accesses. The row buffer interference arises when a non-critical writeback request induces a row buffer conflict with a critical read request.

In particular, we make the following contributions:

1) A novel LLC tag design for small RB organization that significantly improves performance via fast tag lookups compared to existing tag designs [4]–[6], [8].
2) While the tag design provides fast tag lookp it leads to internal fragmentation when applied to hybrid DRAM/STT-

The authors are with the Chair for Compiler Consruction, Technische Universität Dresden, Germany e-mail: {fazal.hameed, jeronimo.castrillon}@tu-dresden.de. Fazal Hameed is also affiliated with Institute of Space Technology, Islamabad, Pakistan.

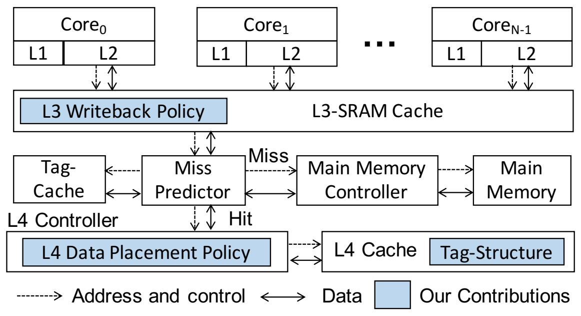


Fig. 1: Typical LLC organization with four-level cache hierarchy employing die-stacked memory as LLC [3], [4]

RAM LLC. To reduce the unused space caused by the internal fragmentation, we provide an improved tag design that intelligently stores the tags and the data.

3) A *Data Placement Policy* for the hybrid LLC that reduces the number of row fetches compared to existing data placement policies.
4) A proactive *L3-SRAM Writeback Policy* that evicts dirty adjacent blocks from L3-SRAM which are mapped to the same LLC row. These dirty blocks are serviced efficiently with much lower latency compared to existing writeback policies by reducing row buffer interference.

This paper is organized as follows. Section II outlines the background and the related work. Section III presents the details of our LLC architecture. The experimental setup and the benchmarks used for the evaluation are discussed in Section IV. Sections V and VI provide qualitative and quantitative comparisons with state-of-the-art approaches followed by conclusions in Section VII.

II. Motivation and Background

This section describes the movitation and introduces the background and the particulars of the most recent tag designs. It also provides the details of state-of-the-art hybrid LLC architectures.

A. Large RB Organization

Typically an LLC (either based on DRAM or STT-RAM) is organized into a number of logically independent banks where each bank consists of multiple rows of data. In a large RB organization [3]–[5], [8], [21], [22], a single large RB is available at each bank. When an access is made to a bank, one row of the bank is fetched into the bank's RB. This operation is called as *row-activate*. Any subsequent access to the row residing in the RB (called RB hit) will not require the row-activate operation. When an access is made to a different row of the same bank (called RB miss), the contents of the RB are substituted by the new row after the current row residing in the RB is written back to the bank. This operation is called as *precharge*. An RB miss requires to store the contents of the current row (precharge operation), time to read the new row (row-activate operation) and the RB read/write access time. On the other hand, an RB hit only requires the RB read/write access time which significantly reduces access latency and energy compared to an RB miss. Removing a row from the

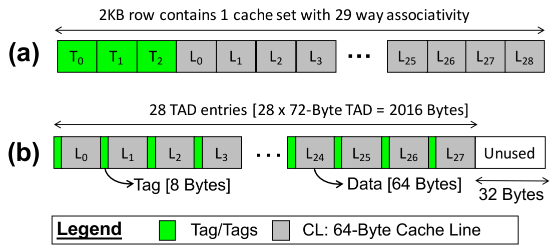


Fig. 2: The row organization of (a) LH-Tag design [5], [6] (b) Direct-mapped Alloy-Tag design [21]; for a 2KB row size

row buffer to accommodate a new row is called *row buffer interference*.

B. Small RB Organization

The small RB organization proposed in [9] employs multiple small RBs per bank instead of a single large RB per bank. Their approach divides the large row and the RB into multiple sub-rows and sub-RBs respectively. They only fetch the requested sub-row into one of the sub-RB instead of fetching the entire row. Compared to large RB organization, the small RB organization significantly improves the energy consumption due to reduction in the energy consumed by the activate and precharge operations on the smaller sub-rows. Additionally, in a large RB organization the entire bank is unavailable while an operation is being performed on the RB, and therefore any access to other rows of the same bank will be delayed until the current operation on the RB completes. Serialized accesses to two different rows of the same bank suppresses the bank-level-parallelism, which increases the queuing delay at the LLC controller. The use of multiple RB's in small RB organization allows multiple accesses to the same bank in parallel which improves bank-level-parallelism, herby reducing the queuing delay.

C. LH-Tag Design

The LH-Tag stores the tags and data of LLC in the same row [5], [6]. The LH-Tag for a 2KB row is shown in Fig. 2-(a). Each 2KB LLC row comprises one cache set with 29-way associativity. The row consists of three tag blocks and 29 cache lines. Once the entire 2KB row is fetched into the RB, the three tag block must be be accessed before the cache line. Both of these operations are directly performed on the RB.

D. Alloy-Tag Design

The direct-mapped Alloy-Tag [21] arranges the tag and data side by side in an LLC row to constitute a single *Tag and Data* (TAD) entry, as shown in Fig. 2-(b). Compared to LH-Tag, the Alloy-Tag leads to reduced LLC hit latency because a single access of the TAD entry is required from the row buffer instead of having isolated accesses for the tag and data. However, this reduction in the hit latency comes at the expense of higher LLC miss rate when compared to LH-Tag. To service an LLC hit after the tag matches, the data field of the corresponding TAD is transferred to the requesting core.

TABLE I: Latency of a read request for different scenarios in state-of-the-art DRAM cache [3], [4], [8] excluding controller latency

RB Hit	Tag-Cache Hit	Tag Check Latency	Cache Line Latency	Total Latency
No	No	24 cycles	40 cycles	64 cycles
Yes	No	24 cycles	22 cycles	46 cycles
No	Yes	2 cycles	40 cycles	42 cycles
Yes	Yes	2 cycles	22 cycles	24 cycles

E. LAMOST-Tag Design with a Tag-Cache

In LAMOST-Tag design, each 4KB LLC row consists of 8 cache sets each set with 7-way associativity [3], [4], [8]. Each set consists of one tag block and seven cache lines as shown in Fig. 4-(b). The authors also propose a small low latency hardware structure namely Tag-Cache that caches the tags of recently accessed LLC sets. Table I shows the latency of a read request for different scenarios in LAMOST-Tag design. Note that the latency values do not show the queuing delay in the LLC controller (time spent in the LLC controller before having an access to an LLC bank). Accesses that hit in the Tag-Cache are serviced with much lower latency because they do not require LLC access for the tags. Similarly, accesses that hit in the RB are serviced with much lower latency because they do not require LLC bank access for the cache line. The performance of a DRAM cache depends upon both RB and Tag-Cache hit rates.

F. State-of-the-art Hybrid LLC Architectures

Until now, small caches have been combined in hybrid LLCs with fast SRAM and slow STT-RAM regions [11]–[16]. These LLC architectures aim to exploit the best of the positive characteristics of both SRAM and STT-RAM technologies. In some of these techniques, the SRAM cache is used to store frequently written data to tackle high STT-RAM write latencies and write energy [11]–[14]. For instance, a migration technique is presented in [13] that migrates high-reused data at runtime from slower STT-RAM regions to faster SRAM regions. The hybrid SRAM/STT-RAM LLC architecture in [16] applies power gating to memory arrays for power reduction while considering the application cache access patterns.

Existing techniques originally proposed for hybrid SRAM/STT-RAM LLC, e.g., in [11]–[14], cannot be directly applied to hybrid DRAM/STT-RAM LLC designs. First, existing designs store the tags of STT-RAM LLC in a separate SRAM tag array which is not a viable option for a larger LLC due to large area and power overheads. Second, existing data placement policies do not consider row buffer interference because smaller SRAM and STT-RAM caches do not require row buffers. It is to expect that these kinds of row-buffer-agnostic policies to degrade performance when applied to hybrid DRAM/STT-RAM LLC. Please note that row buffers are employed for larger memories (i.e. DRAM or STT-RAM [7], [23]–[25]) to hide the long bank access latencies.

G. Challenges of a hybrid LLC architecture

As discussed above, existing hybrid LLC architectures have remained relatively small in the past combining small SRAM and STT-RAM caches. These hybrid LLC architectures have a severe limitation because they can only provide a limited cache capacity from 1 MB to 16 MB. This *Capacity Wall* is particularly critical for emerging applications with large working set sizes, e.g., graphic processing, scientific, and multimedia applications. A larger LLC is required to close the speed gap between processor and off-chip memory for emerging applications.

To address the *Capacity Wall*, previous studies have introduced larger DRAM LLCs [2]–[5] and STT-RAM LLC [7]. However, the DRAM LLC energy consumption has become a significant concern in high-performance computing platforms. For instance, the authors of [26] showed in an energy breakdown of a recent multi-core platform that DRAM LLC contributes around 15-25% to the total system energy consumption.

The high DRAM energy consumption is primarily caused by high refresh energy to avoid data loss in the DRAM cell, whereas STT-RAM-based designs suffer from high write latency and worse endurance [17]–[20]. These conflicting characteristics motivate hybrid designs that overcome the limitations of the single technologies, while still providing the high bandwidth required by emerging application. Key to such a hybrid design is a novel LLC tag design and an architecture that mitigates destructive row buffer interference. These two aspects are the matter of the next section.

III. Proposed LLC Architecture

The high-level overview of our hybrid LLC architecture is shown in Fig. 1. Similar to [9], we employ small RB organization as shown in Fig. 3. The small RB organization has been shown to have performance and energy benefits compared to the large RB organization [3], [4], [8] as explained in Section II-B. However, the main drawback of existing small RB organization is that it suffers from reduced Tag-Cache hit rate. To improve it, we devise the Split-Tag design (cf. Section III-A) depicted in Fig. 3 and Fig. 4-(a). Consequently, we present the details of our hybrid LLC tag design in Section III-B followed by our *Data Placement Policy* in Section III-C. Section III-D explains our *L3-SRAM Writeback Policy* to reduce row buffer interference by taking the LLC characteristics into account.

A. Split-Tag Design

The high level view of the Split-Tag design is illustrated in Fig. 3 where each bank contains 8 subarrays, i.e., Subarray$_0$ to Subarray$_7$, with Subarray$_0$ dedicated for storing the tags. A logical row size of 4 KB includes 8 sub-rows spread across 8 subarrays, each sub-row containing 512 bytes, i.e., eight 64-byte blocks. With Subarray$_0$ reserved for the tags, each sub-row accommodates an 8-way associative set. The tag block (shown in green box in Fig. 4-a) stores the tags of all cache lines within a set. In contrast, LAMOST-Tag design [4], [8],

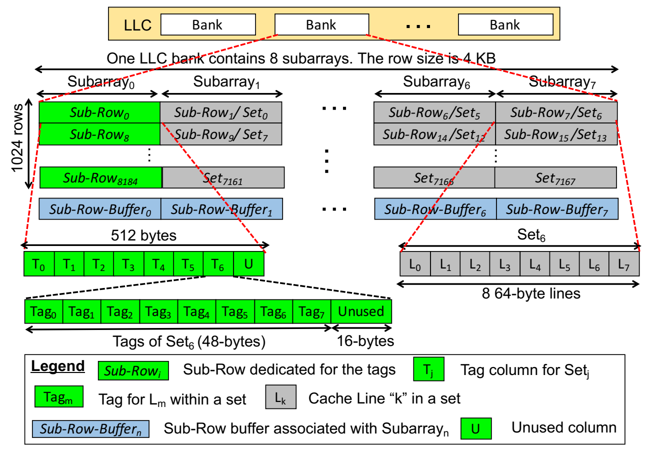


Fig. 3: High level view of the Split-Tag design

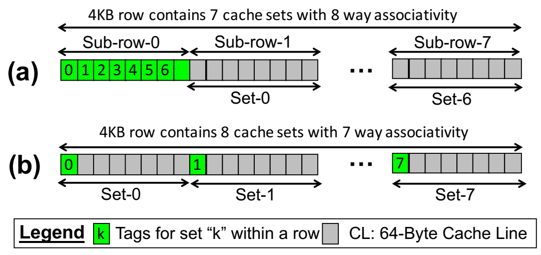

4KB row contains 7 cache sets with 8 way associativity
Sub-row-0
Sub-row-1
Sub-row-7
(a)
0 1 2 3 4 5 6
Set-0
Set-6
4KB row contains 8 cache sets with 7 way associativity
(b)
0
1
7
Set-0
Set-1
Set-7
Legend
k Tags for set "k" within a row
CL: 64-Byte Cache Line

Fig. 4: The row organization of (a) Our Split-Tag desing (b) LAMOST-Tag design [4], [8], [22]

[22] stores the tag block with in the same sub-row along with the cache lines as shown in in Fig. 4-(b).

An access to a cache line in our Split-Tag requires two sub-rows accesses (one for the tag block and other for the cache line access) in contrast to a single sub-row access in LAMOST-Tag (tag block and the cache line reside in the same sub-row). To reduce the number of accesses to the tags in the bank, we employ a small low-latency SRAM-based Tag-Cache similar to [3], [4], [8] that exploits the temporal locality by caching the tags of spatial adjacent LLC sets. Therefore, a Tag-Cache hit bypasses the sub-row access dedicated for the tags while reading the tags directly from the Tag-Cache. Let us assume that there is an LLC read request to a cache line that belongs to Set-6 in Fig. 4-(a). To elaborate how our approach works along with the Tag-Cache, we describe the implementation of the following important events to service the above request:

Tag-Cache miss: On a Tag-Cache miss, the sub-row dedicated for the tag blocks (i.e. Sub-row-0) is accessed to read the requested tag block (i.e. tag block ”6”). This tag block indicates the location of the cache line in Set-6 (stored in Sub-row-7). After reading the tag block, the LLC controller issues a read request to access the requested cache line in Sub-row-7 (i.e. Set-6) which is forwarded to the requesting core. After that, subsequent read requests are sent to access the remaining tag blocks (i.e. tag blocks ”0” to ”5”) from Sub-row-0 which are placed in the Tag-Cache. We exploit the temporal locality of applications that these prefetched tag blocks will likely be accessed in the near future. In contrast, using LAMOST-Tag design [4], [8], [22] employing small RB organization [9] need to fill only the requested tag block (i.e. tag block ”6”) in the Tag-Cache. The reason is that accessing adjacent tag blocks

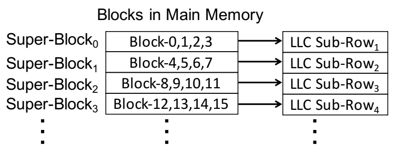


Fig. 5: Memory block to sub-row mapping in the Split-Tag design

will require multiple sub-row accesses (see Fig. 4-b) which is not a viable option in the small RB organization.

Tag-Cache hit: A Tag-Cache hit does not require any LLC lookup for the tag block (i.e. it is directly accessed from the Tag-Cache), so a read request is directly sent to Sub-row-7 to access the requested cache line in Set-6. The tag block ”6” is updated in the Tag-Cache (e.g. to update LRU and dirty information etc.) but not in the LLC. In contrast to our approach, existing tag designs [4], [8] always update the tag block both in the Tag-Cache and the LLC. In our implementation, the tags in the LLC are only updated after its eviction from the Tag-Cache.

RB miss: If there is an RB miss for the sub-row (i.e. Sub-row-0) containing the tag block, then it is fetched into one of the small sub-RBs available at the LLC bank shown in Fig. 3. Note that Sub-row-0 only needs to be accessed after a Tag-Cache miss. If the sub-row containing the requested cache line (i.e. Sub-row-7 in the above example) does not reside in the RB, then it is fetched into one of the small sub-RB’s. In contrast to our approach, state-of-the-art large RB organization [3]–[5], [7], [8], [22] fetches all sub-rows (i.e. Sub-row-0 to Sub-row-7) into the the large RB after an RB miss. Our proposed approach only fetches the requested sub-rows instead of fetching all sub-rows that provides significant energy saving compared to the large RB organization. On the other hand, the small RB organization [9] only fetches one sub-row after an RB miss because the tag block and the cache line resides in the same sub-row (see Fig. 4-b). Compared to the small RB organization, the additional latency incurred in fetching the sub-row dedicated for the tag-blocks in our proposed approach is compensated by significant improvement in the Tag-Cache hit rate (see Section V-A and Table V for evaluation).

Fig. 5 illustrates how the Split-Tag design maps memory blocks to sub-rows. This design assigns 4 consecutive memory blocks to the same LLC sub-row. For instance, memory blocks 0-3, 4-7, 8-11, and 12-15 are assigned to LLC Sub-Row$_1$, Sub-Row$_2$, Sub-Row$_3$, and Sub-Row$_4$ respectively. For the rest of the paper, four consecutive memory blocks are referred to as a super-block (see Fig. 5). This mapping exploits the spatial locality of applications, i.e., adjacent memory blocks are likely accessed together, thereby improving the row buffer hit rate. Note that each LLC set is stored in a separate sub-row as shown in Fig. 3 and Fig. 4-(a).

1) Internal fragmentation: In the Split-Tag design, the sub-rows in Subarray$_0$ contains 7 tag blocks, i.e., T_0 to T_6, with one 64-byte block left unused. In each tag block, 16 bytes are not used, for a total of 112 unused bytes. As a result, the total unused space in the 4 KB row is 176 bytes. This unused space leads to internal fragmentation in the LLC. Each physical row

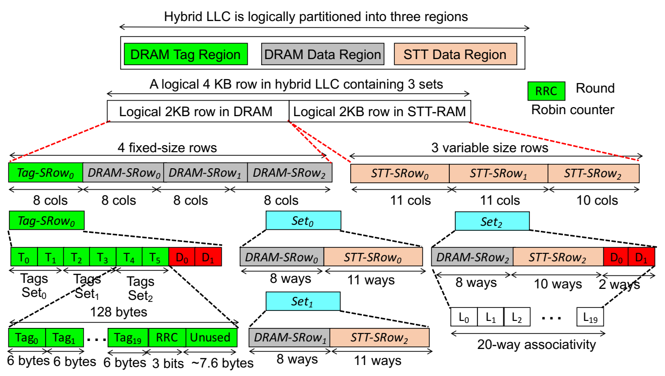


Fig. 6: High level overview of our Hybrid-Split-Tag LLC design.

in Subarray$_1$ to Subarray$_7$ accommodates a single set.

B. *Hybrid-Split-Tag*

As mentioned above, Split-Tag design incur internal fragmentation due to two reasons. First, 16 bytes are left unused in each tag block of Subarray$_0$ (i.e. dedicated for storing the tags). Second, one entire 64-byte block is left unused in Subarray$_0$. Our Hybrid-Split-Tag design mitigates the former problem by packing more tag entries in the tag blocks (referred to as *Rule-1*). The later problem is addressed by storing additional ways in the unused blocks of Subarray$_0$ (referred to as *Rule-2*).

Fig. 6 gives an overview of our Hybrid-Split-Tag design. As shown, the hybrid LLC is logically organized into three regions called DRAM-Tag-Region, DRAM-Data-Region and STT-Data-Region. A logical 4 KB row in our Hybrid-Split-Tag LLC comprises one logical 2 KB DRAM row and one logical 2 KB STT-RAM row. The logical 4 KB row consists of three LLC sets namely Set$_0$-Set$_2$. The logical 2 KB DRAM row consists of 4 fixed size sub-rows each containing eight blocks. The first sub-row, i.e., Tag-SRow$_0$, stores the tags of the three sets. The next three rows, i.e., DRAM-SRow$_0$-DRAM-SRow$_2$, store the data. The logical 2 KB STT-RAM row consists of 3 unequal size sub-rows where the first two sub-rows contain 11 blocks and the third one contains 10 blocks. The tag block pairs T$_0$-T$_1$, T$_2$-T$_3$, and T$_4$-T$_5$ of Tag-SRow$_0$ store the tags of Set$_0$, Set$_1$, and Set$_2$ respectively. Set$_0$ and Set$_1$ provide 19-way associativity with 8 and 11 ways stored in a DRAM sub-row and an STT-RAM sub-row respectively. Set$_2$ provides 20-way associativity where 8 and 10 ways are stored in a DRAM sub-row (in DRAM-SRow$_2$) and an STT-RAM sub-row (in STT-SRow$_2$) respectively. The remaining two ways (D$_0$ and D$_1$) are stored in Tag-SRow$_0$.

A request to the hybrid LLC set requires an access to the two tag blocks to identify (a) an LLC hit/miss and (b) the location of the cache line, i.e., whether in the DRAM-Data-Region, in the DRAM-Tag-Region (last two blocks of Tag-SRow$_0$ for Set$_2$), or in STT-Data-Region. In Hybrid-Split-Tag 35 bytes are left unused in each logical 4 KB row as opposed to 176 bytes in the Split-Tag (see Section III-A). Thus, our proposal reduces the waste of space due to DRAM internal fragmentation. This savings are made possible by packing more tag entries in the tag blocks. Also, the last two blocks in Tag-SRow$_0$ are used to store two ways of Set$_2$. The efficient storage utilization and high LLC associativity of our Hybrid-Split-Tag design provides reduced LLC miss rate compared to Split-Tag design as will be discussed in Section VI-D (Fig. 21b).

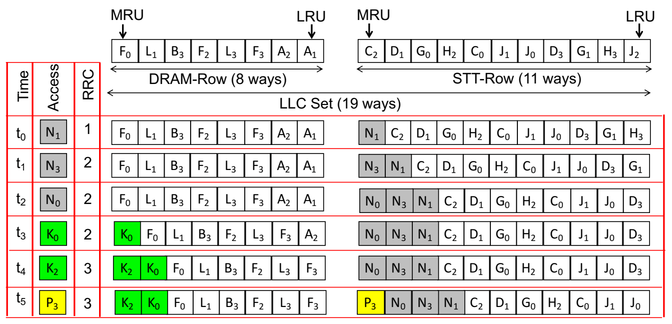


Fig. 7: Example of *Data Placement Policy* (MRU: Most Recently Used, LRU: Least Recently Used)

Our hybrid LLC tag design is based on the notion of the above-mentioned two rules (i.e., *Rule-1* and *Rule-2*). Fig. 6 shows our Hybrid-Split-Tag desig for a logical 4 KB row size. However, this design is generic and the same rules can be applied to other logical row sizes as well.

C. *Data Placement Policy*

1) Reducing the number of LLC row fetches: The hybrid LLC architecture has to decide which memory to use after an LLC miss, i.e., DRAM or STT-RAM. We first present a motivating example in Fig. 7 that shows how our *Data Placement Policy* reduces the number of row fetches. This example shows a 19-way set of our hybrid LLC mentioned in Section III-B. For this example we assume that:

1) The set in Fig. 7 initially contains arbitrary blocks from many super blocks (i.e. from SB_A to SB_L).
2) A super block consists of 4 consecutive blocks. For instance, the super block SB_N consists of four consecutive blocks N_0-N_3. Similarly, the super block SB_K contains K_0-K_3.
3) Every block of a super block is mapped to the same LLC set which consists of a DRAM sub-row and an STT sub-row as shown in Fig. 7.
4) The blocks N_1, N_3, and N_0 of a super block SB_N are requested at time t_0, t_1, and t_2 respectively and are currently absent in LLC.

Existing data placement policies [13], [14] may place some blocks (e.g., N_1) of the super block SB_N in the DRAM sub-row and other blocks (e.g., N_3 and N_0) in the STT sub-row. In this scenario, two LLC rows need to be fetched in the row buffer which exacerbates the energy consumption. Also, fetching two rows necessitate to evict two victim rows from the row buffer which deteriorates the row buffer hit rate. This is due to the fact, as mentioned before, that these policies were originally proposed for smaller hybrid SRAM/STT-RAM cache architectures without a row buffer. In our proposal, all blocks belonging to a super block are either placed in a DRAM sub-row or in an STT sub-row. For the example in Fig. 7, the

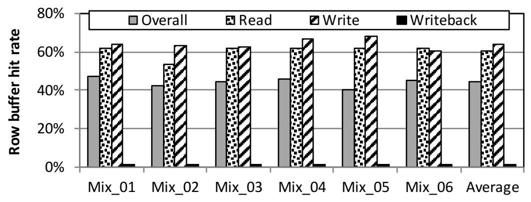


Fig. 8: Row buffer hit rates of different SPEC2006 applications (see Table II) for LAMOST-Tag design with 4KB row buffer size employing 256 MB DRAM LLC

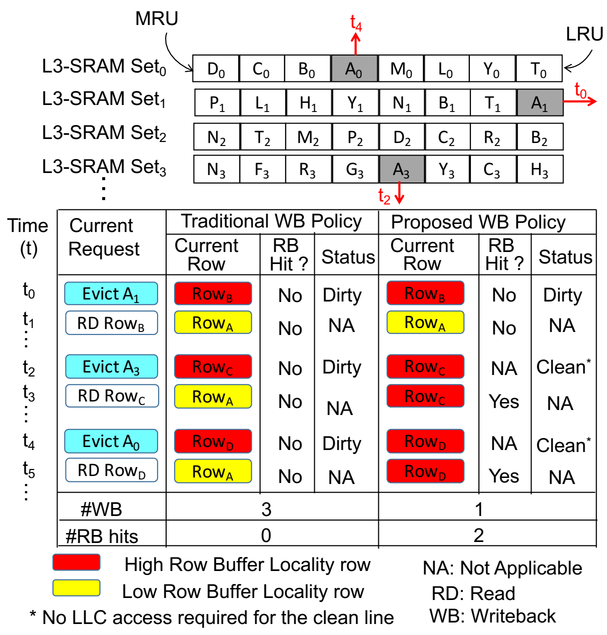
blocks N_1, N_3, and N_0 are placed in an STT sub-row as they belong to the same super block SB_N. Thus, our architecture need to only fetch a single LLC row to place or access adjacent blocks of a super block.

2) Providing balanced set utilization: Besides reducing the number of LLC row fetches, another goal of our *Data Placement Policy* is to uniformly distribute blocks to DRAM-Data-Region and STT-Data-Region. This goal is achieved by assigning more blocks to STT-RAM because an STT sub-row has 3 more ways compared to the DRAM sub-row. To this end, we employ a saturating 3-bit **R**ound **R**obin **C**ounter (RRC) for each LLC set (see Fig. 6). To illustrate our policy, we assume that a block $B_{i:i\in\{0,1,2,3\}}$ belonging to a super block S is absent in the LLC. The adjacent blocks of the missing block B_i are searched in the relevant set after an LLC miss. If any of the adjacent blocks is found, then B_i is placed in a row of the same memory type as the adjacent blocks. This approach reduces the number of LLC row fetches as illustrated earlier. The RRC remains unchanged if the adjacent block of B_i is found in the LLC. If an LLC miss occurs for the super block S, i.e., B_0-B_3 are absent in LLC, then B_i is placed in a DRAM sub-row if the previous value of RRC is 2, 5 or 7. Otherwise, B_i is placed in an STT sub-row of the relevant LLC set, i.e., with an RRC of 0, 1, 3, 4 or 6. The RRC is incremented when the super block S misses the LLC.

The set balancing using our *Data Placement Policy* is illustrated with the example in Fig. 7. Block N_1 is placed in STT sub-row at t_0 because the previous value of RRC is 1 and adjacent blocks of N_1 are absent in LLC, i.e. an LLC miss occurs for the super block SB_N. The RRC is incremented after the insertion of block N_1 as a result of an LLC miss for the super block SB_N. The following missing blocks N_3 and N_0 are placed in an STT sub-row (since N_1 was assigned to an STT sub-row as well) while the RRC remains unchanged. Similarly, block K_0 is placed in DRAM sub-row at t_3 because the previous value of RRC is 2 and an LLC miss occurs for the super block K.

D. L3-SRAM Writeback Policy

The different LLC accesses, i.e., read, write, and writeback, have varying row buffer hit rates as can be observed in Fig. 8. Writeback accesses have a low row buffer hit rate of less than 2% and are classified as low **R**ow **B**uffer **L**ocality (RBL) accesses. These low RBL writeback accesses cause eviction of high RBL rows, i.e., rows with frequent row buffer hits, from the row buffer. Therefore, the dirty evicted L3-SRAM


MRU
LRU
t_4
t_0
t_2
L3-SRAM Set$_0$ D_0 C_0 B_0 A_0 M_0 L_0 Y_0 T_0
L3-SRAM Set$_1$ P_1 L_1 H_1 Y_1 N_1 B_1 T_1 A_1
L3-SRAM Set$_2$ N_2 T_2 M_2 P_2 D_2 C_2 R_2 B_2
L3-SRAM Set$_3$ N_3 F_3 R_3 G_3 A_3 Y_3 C_3 H_3

Time (t)	Current Request	Traditional WB Policy			Proposed WB Policy		
		Current Row	RB Hit ?	Status	Current Row	RB Hit ?	Status
t_0	Evict A_1	🟥 Row_B	No	Dirty	🟥 Row_B	No	Dirty
t_1	RD Row_B	🟨 Row_A	No	NA	🟨 Row_A	No	NA
t_2	Evict A_3	🟥 Row_C	No	Dirty	🟥 Row_C	NA	Clean*
t_3	RD Row_C	🟨 Row_A	No	NA	🟥 Row_C	Yes	NA
t_4	Evict A_0	🟥 Row_D	No	Dirty	🟥 Row_D	NA	Clean*
t_5	RD Row_D	🟨 Row_A	No	NA	🟥 Row_D	Yes	NA
	#WB	3			1		
	#RB hits	0			2		

🟥 High Row Buffer Locality row
🟨 Low Row Buffer Locality row
* No LLC access required for the clean line
NA: Not Applicable
RD: Read
WB: Writeback

Fig. 9: Example illustrating traditional and proposed L3-SRAM writeback policy

lines are the major source of row buffer interference, thereby reducing the overall row buffer hit rate.

Our *L3-SRAM Writeback Policy* is based on the notion of diminishing the penalty of dirty L3-SRAM line eviction by taking the LLC characteristics into account. We illustrate the LLC and row buffer interference caused by dirty L3-SRAM lines using a simple example with four L3-SRAM adjacent sets, as shown in Fig. 9. We make the following assumptions: (a) A super block SB_A contains four adjacent memory blocks A_0-A_3 which are assigned to L3-SRAM Set$_0$-Set$_3$ respectively. (b) Blocks A_0, A_1, and A_3 (grey boxes in Fig. 9) are currently present in L3-SRAM in dirty state. (c) The four blocks from super block SB_A are mapped to the same LLC row namely Row_A (yellow box in Fig. 9). (d) Row_A is a low RBL row because it is accessed by writeback requests.

The table in Fig. 9 presents a sample sequence of dirty block evictions at different time steps in traditional policies [2], [7], [21] compared to our proposed policy. This example shows that the dirty blocks A_1, A_3, and A_0 are evicted from L3-SRAM at time t_0, t_2, and t_4 respectively. It also shows that high RBL rows Row_B, Row_C, and Row_D (shown in red boxes) initially reside in the row buffer at t_0, t_2, and t_4 respectively. In the traditional policy, the dirty L3-SRAM writeback accesses (i.e., A_1, A_3, and A_0) cause eviction of the high RBL rows (i.e. Row_B, Row_C, and Row_D) from the row buffer, thereby causing row buffer interference. Any subsequent access to the high RBL row results in a row buffer miss which degrades the performance and energy efficiency. Therefore, the traditional policy leads to a reduced number of row buffer hits as illustrated in Fig. 9.

In our proposed *L3-SRAM Writeback Policy*, whenever a dirty block of a super block is evicted from an L3-SRAM set, it searches for its adjacent dirty blocks in the neighbouring sets. All adjacent dirty blocks of a super block are then written

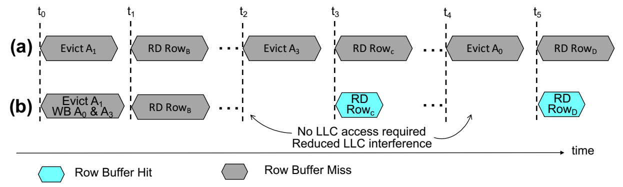


Fig. 10: Service timeline of requests for the example in Fig. 9 employing (a) traditional (b) proposed; L3-SRAM writeback policy

back to LLC as a single request. In the above example, when the dirty block A_1 is evicted from L3-SRAM at t_0, its adjacent dirty blocks A_3 and A_0 are also written back to the LLC in a proactive fashion. This changes the cache state of A_3 and A_0 from dirty to clean at t_0. Note that A_3 and A_0 are not evicted from L3-SRAM at t_0. They are evicted when they are replaced by new cache lines in their respective sets. The eviction of blocks A_3 (at t_2) and A_0 (at t_4) from L3-SRAM do not require an LLC access because they are already written back to LLC. Our *L3-SRAM Writeback Policy* significantly reduces LLC interference by reducing the number of LLC writeback requests. For the example in Fig. 9, the number of writeback requests are reduced from 3 to 1 compared to the traditional policy. Our proposal provides improved row buffer hit rate by mitigating the negative impact of writeback accesses. For the example in Fig. 9, our proposal provides 2 row buffer hits compared to none in the traditional policy. It is worth to mention that dirty writebacks A_1, A_3, and A_0 are serviced back-to-back on the same row buffer (i.e. they are mapped to Row_A), thereby resulting in significant LLC service time reduction. Fig. 10-(a) and Fig. 10-(b) show the timeline of the example requests in Fig. 9 being serviced by the traditional and our proposed L3-SRAM writeback policy respectively. Fig. 10 clearly highlights the problem of increased LLC interference and reduced row buffer hits in the traditional policy compared to our policy.

E. Comparison with state-of-the-art L3-SRAM Writeback Policy

Our L3-SRAM writeback policy is similar to existing L3-SRAM writeback policies [27], [28]. All these policies perform early writeback of dirty blocks which are mapped to the same LLC row. However, for performance benefits, our proposal has a different L3-SRAM set mapping policy which is highlighted in Fig. 11. Existing L3-SRAM set mapping policies map all four blocks of the super block to the same L3-SRAM set as shown in Fig. 11-(a). The primary disadvantage of their approach is that they incur high L3-SRAM miss rate (Fig. 22; Section VI-E). The L3-SRAM miss rate is exacerbated due to within-set contention since all four blocks of the same super block are mapped the same L3-SRAM set. Our L3-SRAM set mapping policy in Fig. 11-(b) eliminates within-set contention because spatially adjacent blocks of a particular super block are mapped to different L3-SRAM sets.

Our L3-SRAM set mapping policy requires four set comparisons instead of a single set comparison before performing a writeback operation. These four set comparisons are required

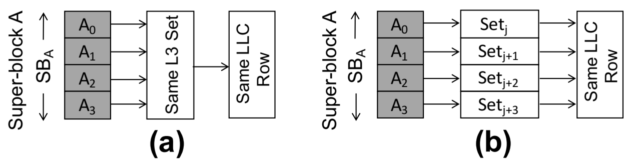


Fig. 11: For super block SB_A comprising 4 blocks (a) State-of-the-art [27], [28] L3-SRAM set mapping (b) our L3-SRAM set mapping

TABLE II: Benchmarks mixes from SPEC2006. LLC-intensive applications shown in italic. LLC-Mem-Sensitive applications shown in non-italic

Mix_01	soplex.r, *bzip*, leslie3d.r, libquantum milc, *astar.t*, *leslie3d.t*, *bzip*, omnetpp
Mix_02	leslie3d.r(2), *astar.t*(2), lbm(2), libquantum(2)
Mix_03	*leslie3d.t*(2), *bzip*(2), omnetpp, milc, lbm, soplex.r
Mix_04	*astar.t*, leslie3d.r, milc, omnetpp(2), soplex.r(2), *leslie3d.t*
Mix_05	leslie3d.r(2), milc(2), *astar.t*, lbm, libquantum, *bzip*
Mix_06	libquantum, soplex.r, omnetpp(2), lbm, *leslie3d.t*(2), *bzip*

to locate the adjacent dirty blocks (if present in L3-SRAM) for a writeback request. It is worth to mention that an L3-SRAM read/write hit request only require a single set comparison. In our and existing implementations, the read and write hit requests are prioritized over writeback requests. Therefore, the three additional set comparisons for writeback requests does not incur a performance penalty. The reason is that writeback requests are non-critical and are performed off the critical path. As a result, they do not effect the latency of critical read requests.

IV. Experimental Setup

We evaluate our policies using the cycle-accurate multi-core Zesto simulator [29]. We model eight out-of-order cores where each core runs a single application. All application mixes constituted from the SPEC CPU2006 benchmark suite [1], [30] are listed in Table II. These application mixes were chosen because they contain applications with different working set sizes and cache access patterns. For instance, Mix_03 and Mix_06 have low LLC miss rates compared to other application mixes. We categorize the applications into *LLC-sensitive* and *LLC-Mem-Sensitive* applications. The LLC-sensitive applications

TABLE III: Configuration details as used in the experiments

Core	3.2 GHz, out-of-order, 4-issue
Private L1 Private L2 Shared L3	32 KB, 8-way associativity, 2 cycles latency 512 KB, 8-way associativity, 5 cycles latency 8 MB, 8-way associativity, 20 cycles latency
L4 (DRAM or STT-RAM or Hybrid)	4 channels, 256 MB, 64 banks, 256-bit channel width, 2 cycle bus latency, t_{RCD}-t_{RP}-t_{CAS} = 18-18-18 (CPU cycles)
t_{WR}	18 and 38 cycles for DRAM and STT-RAM respectively
Tag-Cache	38 KB, 2 cycle latency [3], [4], [8]
Miss Predictor	Map-I [21], 256 entries
Main Memory (DRAM)	2 channels, 16 KB row buffer, 64-bit channel width, tRAS-tRCD-tRP-tCAS-tWR = 144-36-36-36-36 (CPU cycles)

TABLE IV: Per-bit energy consumption of memory operations normalized to the energy of accessing the row buffer taken from [23]

Operation	Normalized Energy
DRAM Array Read/Write	1.19
DRAM Precharge	0.39
STT-RAM Array Read	1.08
STT-RAM Array Write	2.83
Row Buffer Access	1.00

TABLE V: Comparisons of different configurations. The red color indicates a bad value for a parameter

Configuration	Tag-Cache Hit Rate	Avg. RB Hit Rate	Avg. Miss Rate	Bank-level-parallelism
LH-Tag-2KB	**6.4%**	**3.6%**	24.3%	**Worst**
LAMOST-Tag-2KB	68.8%	37.1%	26.6%	**Worst**
LAMOST-Tag-4KB	79.2%	44.4%	26.7%	**Worst**
LAMOST-Tag-Small	**45.2%**	36.5%	26.9%	Best
Alloy-Tag-Small	NA	48.9%	**35.7%**	Best
Split-Tag	78.7%	37.4%	24.6%	Best

have a low LLC miss rate and a high LLC access rate. These applications are very sensitive to the LLC read latency and are highlighted in italic in Table II. On the other hand, the performance of LLC-Mem-Sensitive applications depends on LLC read hit latency as well as LLC miss rate. Each core is provided with private L1/L2 caches while L3/L4 caches are shared among the cores (recall Fig. 1). Table III shows a summary of the main architectural parameters of the simulated system. The energy values (Table IV) of DRAM and STT-RAM devices are taken from [23]. For all evaluated configurations, we employ a MAP-I predictor for predicting DRAM cache misses from [21] and Tag-Cache from [3], [4], [8]. The LLC scheduler uses FR-FCFS (First Ready First Come First Serve) policy [31].

V. Evaluating Split-Tag Design

For the evaluation using DRAM Cache, an x86 simulator [29] with cycle accurate DRAM timing model is extended to model our Split-Tag-Design. We compare the following different configurations when applied on top of DRAM cache:

- **LH-Tag-2KB:** The LH-Tag design (cf. Section II-C) with a 2 KB RB size employing large RB organization [5], [6].
- **LAMOST-Tag-2KB and LAMOST-Tag-4KB:** The LAMOST-Tag (cf. Section II-E), which employs a RB size of 2 KB [22] and 4 KB [4], [8]. Both of these configurations employ large RB organization (cf. Section II-A).
- **LAMOST-Tag-Small:** LAMOST-Tag with a small RB organization (cf. Section II-B and Fig. 4-(b)). The size of the DRAM row and the sub-row is assumed to be 4 KB and 512 bytes respectively. The memory block to sub-row mapping is similar to the one shown in Fig. 5.
- **Alloy-Tag-Small:** The direct-mapped Alloy-Tag design (cf. Section II-D) build on top of small RB organization with 512 bytes sub-row size [21].
- **Split-Tag:** Our Split-Tag design build on top of small RB organization (i.e. sub-row size is 512 bytes) which is explained in Section III-A. The memory block to sub-row mapping is similar to the one shown in Fig. 5.

We do not employ small RB organization for LH-Tag because it requires fetching all 4 sub-rows to access a cache set which is not a viable option for small RB organization.

A. Performance Analysis

The main performance results for the evaluated configurations are illustrated in Fig. 12, which depicts the performance speedup normalized to LH-Tag. As shown, our Split-Tag improves the overall performance by 41.3%, 15.4%, (21.8%, 17.7%) and 11.3% compared to LH-Tag, Alloy-Tag, LAMOST-Tag with (2KB, 4KB) RB sizes, and LAMOST-Tag-Small, respectively. The performance of DRAM cache is primarily affected by two metrics namely DRAM cache read hit latency and DRAM cache miss rate (depends upon associativity). The DRAM cache hit latency comprises two components: the DRAM array latency and the queueing delay at the DRAM cache controller. The DRAM array latency strongly relies on the row buffer hit rate (higher is better), Tag-Cache hit rate (higher is better) while the queueing delay is strongly influenced by bank-level-parallelism.

Table V provides a quantitative and qualitative comparison of important parameters for the evaluated configurations. The performance of our approach is primarily enhanced via an improved DRAM cache read hit latency compared to all configurations (except Alloy-Tag) as shown in Fig 13. This is because we simultaneously improve (i.e. close to the best value) all of the important parameters. Although our proposal (8-way associative cache) slightly increases the DRAM cache miss rate compared to LH-Tag (29-way associative cache), but that is compensated by significant reduction in the read hit latency (51.2%). LH-Tag provides the worst hit latency compared to all configurations that is the primary reason for its worst performance. LH-Tag has a low row buffer and Tag-Cache hit rate due to reduced temporal locality as illustrated in Table V. In addition, LH-Tag also suffers from reduced bank-level-parallelism due to the use of large RB organization that further worsens the hit latency. The Split-Tag increases the average DRAM cache read hit latency by 12 CPU cycles compared to direct-mapped Alloy-Tag, but that is offset by significant reduction in LLC miss rate (31.1%). Alloy-Tag slightly improves the performance of LLC-Sensitive applications compared to Split-Tag as depicted Fig. 12(b). The reason is that LLC-Sensitive applications benefit from lower LLC read hit latency and are not affected severely by LLC miss rate. However, Alloy-Tag underperforms for LLC-Mem-Sensitive applications (cf. Fig. 12c) because the performance of these applications severely depends on LLC miss rate.

Compared to all variants of LAMOST-Tag, our Split-Tag provides simultaneous improvement in DRAM cache read hit latency and miss rate. It has a better miss rate compared to all variants of LAMOST-Tag because it provides high associativity (i.e. 8-way) compared to LAMOST-Tag (i.e. 7-way). The hit latency compared to LAMOST with (2KB, 4KB) RB sizes is reduced via an improved bank-level-parallelism (cf. Section II-B) because we employ small RB organization. On the other hand, the hit latency compared to LAMOST-Tag-Small is improved via an enhanced Tag-Cache rate (74.1%

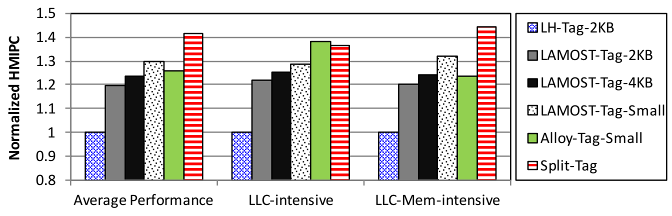


Fig. 12: Performance comparison of different configurations.

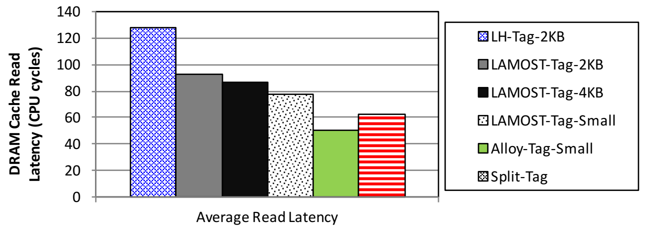


Fig. 13: DRAM cache latency for read request for different configurations

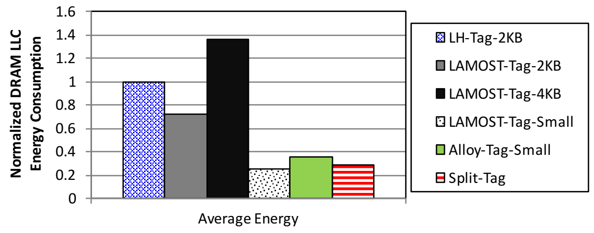


Fig. 14: Energy comparison of our Split-Tag design compared to state-of-the-art approaches

improvement). The reason for this improvement is that our Split-Tag exploits the spatial locality by prefetching all adjacent tag blocks in the Tag-Cache (details in Section III-A). In contrast, LAMOST-Tag-Small only fill the requested tag block in the Tag-Cache.

B. Energy Analysis

The results of our energy analysis are summarized in Fig. 14. As shown, reducing the RB size significantly reduces DRAM cache energy consumption due to reduction in the energy required for the row-activate and precharge operations. The DRAM cache energy savings of our proposal using small RB organization are 71%, 60%, and 78% compared to LH-Tag-2KB, LAMOST-Tag-2KB and LAMOST-Tag-4KB, respectively, because they employ large RB organization. Compared to Alloy-Tag-Small, the energy saving translates to 19% via reduced LLC miss rate (cf. Table V). Our Split-Tag slightly increases the energy consumption compared to LAMOST-Tag-Small. However, this slight increase in the energy consumption is compensated by 11.3% improvement in performance. The reason for this increase is that our Split-Tag requires additional sub-row accesses for the tag block because the tag block and the cache line resides in different sub-rows. However, caching the tag blocks in the Tag-Cache bypasses any future sub-row accesses for the tags while reading the tags directly from the Tag-Cache.

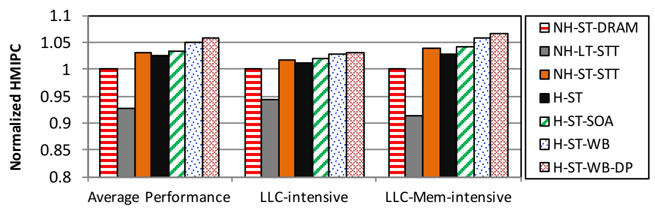


Fig. 15: Normalized Harmonic Mean Instruction per Cycle (HMIPC) results.

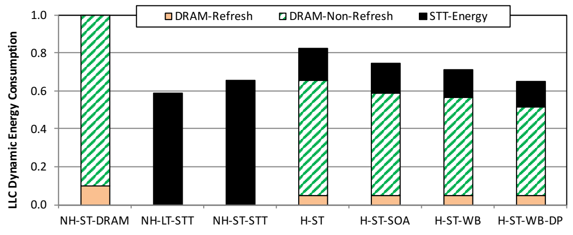


Fig. 16: Normalized energy results for the evaluated architectures.

VI. Evaluating Hybrid-Split-Tag design

This section describes the benefits of our Hybrid-Split-Tag design by evaluating the following LLC architectures:

- **NH-ST-DRAM & NH-ST-STT:** The best-performing non-hybrid (NH) Split-Tag design using DRAM and STT cache respectively with 256 MB LLC.
- **NH-LT-STT:** Existing non-hybrid 256 MB STT-RAM LLC [7] employing LAMOST Tag-Design and STT-RAM specific optimizations in [23].
- **NH-LT-STT:** The non-hybrid 256 MB STT-RAM LLC [7] employing LAMOST Tag-Design and STT-RAM specific optimizations in [23].
- **H-ST:** The hybrid 256 MB LLC comprising 128 MB DRAM and 128 MB STT-RAM with our Hybrid-Split-Tag design elaborated in Section III-B. This configuration models the state-of-the-art data placement policy in [13], [14] and STT-RAM specific optimizations in [7], [23].
- **H-ST-SOA:** The Hybrid-Split-Tag design extended by state-of-the-art (SOA) L3-SRAM writeback policy [27], [28] highlighted in Fig. 11-(a) and described in Section III-E.
- **H-ST-WB:** The Hybrid-Split-Tag design extended by our L3-SRAM writeback policy (Fig. 11-b) from Section III-D.
- **H-ST-WB-DP:** The H-ST-WB configuration extended by our data placement policy in Section III-C.

We make the following assumptions for all evaluated architectures:

1) We use small RB organization (cf. Section II-B) as in [2], [9], [10] for energy reduction and Tag-Cache for latency reduction.
2) To take into account high write latency for STT-RAM similar to [23], we assume that writes to the STT-RAM array takes 20 cycles more compared to a DRAM array. This implies that t_{WR} (write recovery time) is 18 cycles for DRAM and 38 cycles for STT-RAM.

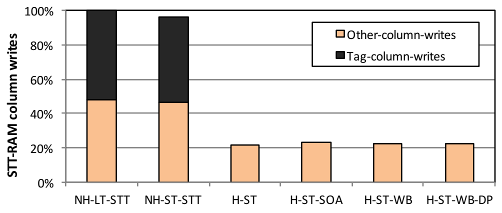


Fig. 17: Number of STT-RAM block writes operations normalized to NH-LT-STT.

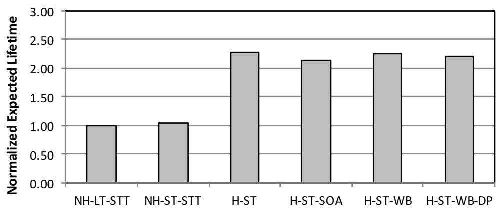


Fig. 18: Normalized expected lifetime results for the evaluated configurations.

A. Experimental Results

Fig. 15 shows the performance comparisons of different evaluated architectures normalized to Split-Tag design applied to DRAM (NH-ST-DRAM). As illustrated, our combined policies (H-ST-WP-DP) provides a Harmonic Mean Instruction per Cycle (HMIPC) speedup of 5.8% and 14.1% compared to Split-Tag (NH-ST-DRAM) and existing STT-RAM LLC (NH-LT-STT) respectively. The energy and the endurance results are depicted in Fig. 16 and Fig. 17 respectively. The energy consumption is reduced by 35% compared to NH-ST-DRAM and the average number of STT-RAM block writes are reduced by 78% compared to NH-LT-STT. The qualitative and quantitative comparison of various architectures is described in the following.

B. Comparing non-hybrid DRAM and STT-RAM architectures

By leveraging the STT-RAM non-volatility characteristics, the non-hybrid STT-RAM applies row buffer bypass optimization [7], [23] which improves the LLC row buffer hit rate from 37.4% (NH-ST-DRAM) to 47.7% (NH-LT-STT). Despite the improvement in the LLC row buffer hit rate, existing non-hybrid STT-RAM LLC worsens the performance by 7.2% compared to our Split-Tag design (NH-ST-DRAM). The positive impact of the increase in the row buffer hit rate in STT-RAM LLC is compensated by reduced Tag-Cache hit rate and high STT-RAM write latency. The Tag-Cache hit rate is worsened compared to our Split-Tag design because existing STT-RAM employs LAMOST-Tag design (cf. Section III-A for details Section V-A for evaluation). Compared to non-hybrid DRAM, the STT-RAM LLC reduces the energy consumption by 41% due to increased row buffer hit rate, the absence of refresh energy and the partial write optimization [23].

C. Impact of Split-Tag design on STT-RAM

Our Split-Tag design is generic and can be applied to STT-RAM as well. By applying non-volatility characteristics, row

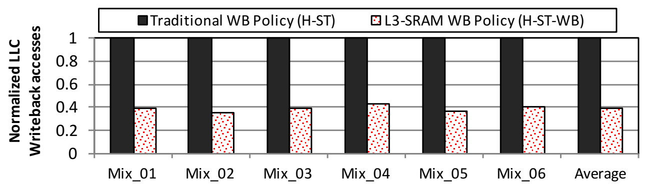


Fig. 19: Total number of LLC writeback accesses normalized to traditional L3-SRAM Writeback policy

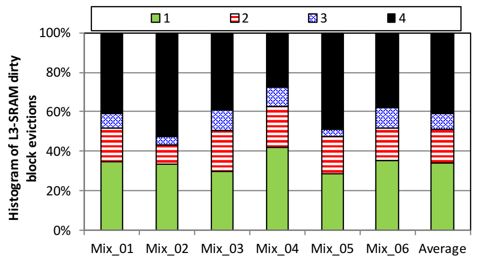


Fig. 20: Distribution of L3-SRAM dirty block evictions within a super block

buffer bypass optimizations in [7], [23], and Split-Tag design on top of STT-RAM, the performance improvement of NH-ST-STT translates to 3.1% compared to NH-ST-DRAM. However, this performance improvement comes at the cost of worst expected lifetime (cf. Section VI-D) compared to NH-ST-DRAM and small increase in energy consumption compared to NH-LT-STT (cf. Fig. 16). The slight energy increase is due to more sub-row read accesses (cf. Section V-B).

D. Impact of Hybrid-Split-Tag design (H-ST)

While STT-RAM outperforms DRAM in terms of energy consumption, the major drawback of existing STT-RAM is its worse expected lifetime. The low expected lifetime is primarily caused by storing the LLC tags in STT-RAM. By storing the tags in DRAM, our Hybrid-Split-Tag significantly reduces the number of STT-RAM block writes by 78% (Fig. 17). As a consequence, the normalized expected lifetime is improved by 2.3× as shown in Fig. 18. The normalized expected lifetime in Fig. 18 is calculated using the following equation [32].

$$NE_{lifetime} = \frac{\#Writes\ in\ baseline \times Memory\ size}{Memory\ size\ of\ baseline \times \#Writes} \tag{1}$$

Additionally, Hybrid-Split-Tag provides improved speedup compared to the non-hybrid DRAM (2.5%) and NH-LT-STT (9.8%) architectures. This speedup is achieved via reduced LLC miss rate (Fig. 21b) due to high LLC associativity (i.e., 19-ways compared to 8-ways) and efficient storage utilization (cf. Section III-B).

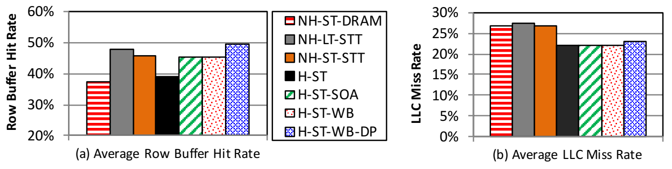


Fig. 21: (a) Average row buffer hit rate (b) Average LLC miss rate; averaged over all benchmark mixes.

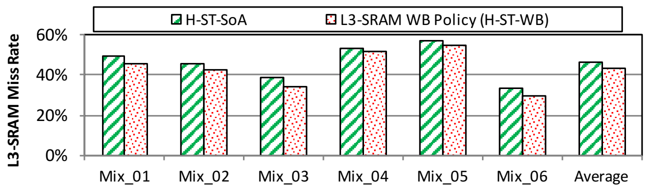


Fig. 22: L3-SRAM miss rate results

E. Impact of L3-SRAM Writeback Policy (H-ST-WB)

With the introduction of the *L3-SRAM Writeback Policy* (Section III-D), the LLC experiences a performance improvement of 2.4% (Fig. 15). The performance improvement compared to the traditional writeback policy is achieved by a 60% reduction in the number of LLC writeback accesses (Fig. 19) and a higher row buffer hit rate (Fig. 21a). The determining factor in the reduction of LLC writeback accesses is the proactive eviction of multiple dirty blocks from L3-SRAM cache. These multiple dirty blocks are serviced as a single request instead of multiple isolated requests. Fig. 20 shows the histogram of L3-SRAM dirty block evictions belonging to a super block for different application mixes. As demonstrated, the multiple dirty block evictions (i.e. 66%) from L3-SRAM are significant compared to a single dirty block eviction (i.e. 34%). By reducing the intensity of low row buffer locality writeback accesses, the row buffer interference is reduced. This leads to an improvement in the row buffer hit rate from 38.9% (without writeback policy) to 45.4% (with writeback policy). This results in energy reduction of 13.8% as shown in Fig. 16.

F. Comparison with state-of-the-art L3-SRAM writeback policy

This section quantitatively and qualitatively analyzes the performance benefits of our approach compared to contemporary L3-SRAM writeback policies [27], [28]. As shown, our L3-SRAM writeback policy (H-ST-WB) improves the overall performance by 1.6% and saves energy consumption by 4.7% compared to existing L3-SRAM writeback policy (i.e. H-ST-SOA). This performance and energy improvement is primarily driven by 6.5% reduction in L3-SRAM miss rate as illustrated in Fig. 22. The high L3-SRAM miss rate of H-ST-SOA compared to our H-ST-WB is due to within-set contention which is caused by mapping all block of a super-block to the same L3-SRAM set (see Fig. 11-a). In contrast, our writeback policy avoids this within-set contention by mapping adjacent blocks of the super-blocks to different L3-SRAM sets (see Fig. 11-b), thereby reducing the L3-SRAM miss rate.

G. Impact of Data Placement Policy (H-ST-WB-DP)

Although H-ST-WB provides high speedup compared to existing proposals, it suffers from increased energy consumption due to an increased number of LLC row fetches. Applying the proposed *Data Placement Policy* (Section III-C) reduces the number of LLC row fetches by 3.7%. This reduction is caused by placing adjacent blocks of a super block in the same LLC sub-row. In contrast, existing data placement policy may place some blocks of a particular super block in a DRAM sub-row and other blocks in an STT-RAM sub-row [13], [14]. By reducing the number of LLC row fetches, our data placement policy reduces the energy consumption by 8.5%.

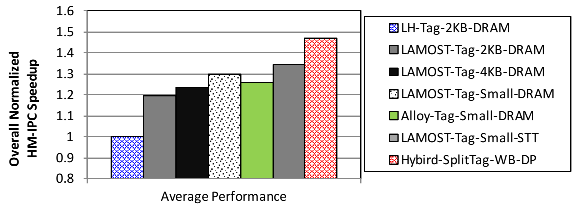


Fig. 23: Performance results showing the impact of our synergistic policies

H. Putting It All Together

This section provides the performance gain of our synergistic policies relative to state-of-the-art non-hybrid DRAM LLC designs evaluated in Section V and STT-RAM LLC design evaluated in this section. Our Hybrid-Split-Tag-WB-DP configuration combines the performance advantages of *Hybrid-Split-Tag* design, *L3-SRAM Writeback Policy* and *Data Placement Policy*. The net performance gain is substantially higher than the performance gain of a single policy which is shown in Fig. 23. On average, Hybrid-Split-Tag-WB-DP improves the average performance by 47.1%, 27.6%, 23.5%, 17.1%, 21.2%, and 13% compared to LH-Tag-2KB-DRAM [5], [6], LAMOST-Tag-2KB-DRAM [22], LAMOST-Tag-4KB-DRAM [4], [8], LAMOST-Tag-Small-DRAM, Alloy-Tag-Small-DRAM [21], and LAMOST-Tag-Small-STT [7] respectively.

The 13% performance improvement compared to contemporary non-hybrid STT-RAM LLC architecture (i.e., LAMOST-Tag-Small-STT) comes at the cost of 9.1% increase in the energy consumption. It is worth mentioning that the energy reduction (24%) and performance increase (17.1%) of the combined policies is prominent compared to existing best-performaning DRAM LLC architecture (i.e., LAMOST-Tag-Small-DRAM).

VII. Conclusions

This paper introduces a novel Split-Tag design for larger LLC that simultaneously improve bank-level-parallelism (via a small RB organization) and Tag-Cache hit rate (via exploiting temporal locality). This design not only retain the energy benefits of small RB organization but it also improves the performance by considering the locality characteristics of applications. Furthermore, we improve the tag design for a hybrid LLC combining DRAM and STT-RAM memory technologies. The proposed design mitigates the STT-RAM endurance issue by storing the tags in DRAM. It also reduces conflict misses via high associativity and efficient storage utilization. We demonstrate that existing LLC architectures suffer from increased LLC interference via increased number of row fetches and writeback accesses. By placing adjacent blocks of a super block in the same LLC row, our *Data Placement Policy* reduces the number of row fetches. In addition, it uniformly distributes data to DRAM and STT-RAM to provide efficient set balancing. To reduce the number of LLC writeback accesses, we propose an *L3-SRAM Writeback Policy* that combines multiple isolated dirty requests into a single request. Simulation results show that our synergistic policies deliver improved performance (17.1%) and energy consumption (24%) compared to the existing best-performing DRAM architecture. It also provides performance speedup of 13% and write endurance improvement of 78% compared to a contemporary STT-RAM architecture.

Acknoledgments

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

References

[1] "Standard Performance Evaluation Corporation," http://www.spec.org, 2017, [Online; accessed 16-May-2018].

[2] F. Hameed *et al.*, "Rethinking On-chip DRAM Cache for Simultaneous Performance and Energy Optimization," in *19th conference on Design, Automation and Test in Europe (DATE)*, March 2017.

[3] C.-C. Huang *et al.*, "ATCache: Reducing DRAM Cache Latency via a Small SRAM Tag Cache," in *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2014, pp. 51–60.

[4] F. Hameed *et al.*, "Architecting On-Chip DRAM Cache for Simultaneous Miss Rate and Latency Reduction," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 651–664, April 2016.

[5] G. Loh *et al.*, "Supporting Very Large DRAM Caches with Compound Access Scheduling and MissMaps," *IEEE Micro Magazine, Special Issue on Top Picks in Computer Architecture Conferences*, pp. 70–78, 2012.

[6] G. Loh *et al.*, "Efficiently Enabling Conventional Block Sizes for Very Large Die-stacked DRAM Caches," in *44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2011, pp. 454–464.

[7] F. Hameed *et al.*, "Architecting STT Last-Level-Cache for Performance and Energy Improvement," in *17th International Symposium on Quality Electronic Design (ISQED)*, March 2016, pp. 319–324.

[8] F. Hameed *et al.*, "Reducing Latency in an SRAM/DRAM Cache Hierarchy via a Novel Tag-Cache Architecture," in *51st Design Automation Conference (DAC'14)*, 2014.

[9] N. D. G. *et al.*, "Multiple Sub-row Buffers in DRAM: Unlocking Performance and Energy Improvement Opportunities," in *26th ACM International Conference on Supercomputing*, 2012, pp. 257–266.

[10] J. Meza *et al.*, "A Case for Small Row Buffers in Non-volatile Main Memories," in *30th International Symposium on Computer Design(ICCD)*, September 2012, pp. 484–485.

[11] I. C. Lin *et al.*, "High-Endurance Hybrid Cache Design in CMP Architecture With Cache Partitioning and Access-Aware Policies," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pp. 2149–2161, Oct 2015.

[12] G. Sun *et al.*, "A Novel Architecture of the 3D Stacked MRAM L2 Cache for CMPs," in *15th International Symposium on High Performance Computer Architecture*, 2009, pp. 239–249.

[13] X. Wu *et al.*, "Hybrid Cache Architecture with Disparate Memory Technologies," in *36th International Symposium on Computer Architecture (ISCA)*, June 2009, pp. 34–45.

[14] X. Wu *et al.*, "Design Exploration of Hybrid Caches with Disparate Memory Technologies," *ACM Transaction on Computer System (TOCS)*, pp. 15:1–15:34, December 2010.

[15] S. Lee *et al.*, "Runtime Thermal Management for 3-D Chip-Multiprocessors With Hybrid SRAM/MRAM L2 Cache," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pp. 520–533, March 2015.

[16] Y. T. Chen *et al.*, "Dynamically Reconfigurable Hybrid Cache: An Energy-Efficient Last-Level Cache Design," in *15th conference on Design, Automation and Test in Europe*, March 2012, pp. 45–50.

[17] Y. Huai, "Spin-Transfer Torque MRAM (STT-MRAM): Challenges and Prospects," *AAPPS Bulletin*, December 2008.

[18] J. Wang *et al.*, "i^2WAP: Improving Non-volatile Cache Lifetime by Reducing Inter- and Intra-set write variations," in *IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, February 2013, pp. 234–245.

[19] P. Wang *et al.*, "Development of STT-MRAM for embedded memory applications," in *2017 IEEE International Magnetics Conference (INTERMAG)*, April 2017, pp. 1–1.

[20] E. Reed *et al.*, "Probabilistic Replacement Strategies for Improving the Lifetimes of NVM-based Caches," in *MEMSYS*, 2017, pp. 166–176.

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

[22] F. Hameed *et al.*, "Simultaneously Optimizing DRAM Cache Hit Latency and Miss Rate via Novel Set Mapping Policies," in *International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES'13)*, 2013.

[23] E. Kultursay *et al.*, "Evaluating STT-RAM as an energy-efficient main memory alternative," in *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2013, pp. 256–267.

[24] F. Hameed *et al.*, "Efficient STT-RAM Last-Level-Cache Architecture to Replace DRAM Cache," in *International Symposium on Memory Systems (MemSys)*, October 2017.

[25] F. Hameed *et al.*, "Performance and energy-efficient design of stt-ram last-level cache," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pp. 1059–1072, June 2018.

[26] E. Vasilakis *et al.*, "Decoupled Fused Cache: Fusing a Decoupled LLC with a DRAM Cache," *TACO*, pp. 65:1–65:23, 2019.

[27] C. J. Lee *et al.*, "DRAM-Aware Last-Level Cache Replacement," 2010.

[28] C. J. Lee *et al.*, "DRAM-Aware Last-Level Cache Writeback: Reducing Write-Caused Interference in Memory Systems," 2010.

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

[30] J. L. Henning, "SPEC CPU2006 Benchmark Descriptions," *SIGARCH Computer Architecture News*, pp. 1–17, September 2006.

[31] S. Rixner *et al.*, "Memory Access Scheduling," in *32nd International Symposium on Computer Architecture (ISCA)*, June 2000, pp. 128–138.

[32] M. K. Qureshi *et al.*, "Enhancing Lifetime and Security of PCM-based Main Memory with Start-gap Wear Leveling," in *42nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2009, pp. 14–23.

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 also 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. Since 2017, Prof. Castrillon is a member of the executive committee of the ACM "Future of Computing Academy".