Improving the Performance of Block-based DRAM Caches via Tag-Data Decoupling

Fazal Hameed, Asif Ali Khan, and Jeronimo Castrillon, *Senior Member, IEEE*

Abstract—In-package DRAM-based Last-Level-Caches (LLCs) that cache data in small chunks (i.e., blocks) are promising for improving system performance due to their efficient main memory bandwidth utilization. However, in these high-capacity DRAM caches, managing metadata (i.e., tags) at low cost is challenging. Storing the tags in SRAM has the advantage of quick tag access but is impractical due to a large area overhead. Storing the tags in DRAM reduces the area overhead but incurs tag serialization latency for an associative LLC design, which is inevitable for achieving high cache hit rate. To address the area and latency overhead problem, we propose a block-based DRAM LLC design that decouples tag and data into two regions in DRAM. Our design stores the tags in a latency-optimized DRAM region as the tags are accessed more often than the data. In contrast, we optimize the data region for area efficiency and map spatially-adjacent cache blocks to the same DRAM row to exploit spatial locality. Our design mitigates the tag serialization latency of existing associative DRAM LLCs via selective in-DRAM tag comparison, which overlaps the latency of tag and data accesses. This efficiently enables LLC bypassing via a novel DRAM Absence Table (DAT) that not only provides fast LLC miss detection but also reduces in-package bandwidth requirements. Our evaluation using SPEC2006 benchmarks shows that our tag-data decoupled LLC improves system performance by 11.7% compared to a state-of-the-art direct-mapped LLC design and by 7.2% compared to an existing associative LLC design.

Index Terms—Die-stacked DRAM Cache, Last Level Cache (LLC), direct mapped cache, associative cache, cache bypassing, metadata management.

1 Introduction

The latency gap between processor and memory has grown exponentially during the past decade. The emergence of multi- and many-core systems has further widened this latency disparity, when cores contend for shared memory. A well-known solution to bridge this latency gap is to integrate multiple in-package DRAM layers on top of the multi-core chip [1]–[5]. These DRAM layers can either be employed as main memory [6], [7] or software managed caches [5], [8]–[10] or LLC [11]–[23]. Employing in-package DRAM as main memory or software managed caches requires re-design of applications and system software; while using it as a hardware managed cache requires no modification to the software stack.

The data in a hardware managed DRAM LLC can be allocated at a small granularity of 64 bytes (referred to as *block-based design*) or at a large granularity of 1 KB/2 KB (referred to as *page-based design*). Page-based LLCs leverage spatial locality by caching large chunks of data in the DRAM LLC but they suffer from over-fetching [5], [8]–[10]. Each LLC fill requires transferring a complete page from the off-package DRAM to the in-package LLC. For pages with limited spatial locality, this results in significant waste of both in-package and off-package bandwidth.

- *A. A. Khan, and J. Castrillon are with the Chair for Compiler Construction, Technische Universität Dresden, 01069 Dresden, Germany. E-mails: {asif_ali.khan, jeronimo.castrillon}@tu-dresden.de*
- *F. Hameed is with the Department of Electrical Engineering, Institute of Space Technology, 44000 Islamabad, Pakistan. Email: fazal.hameed@ist.edu.pk*

Block-based LLCs overcome the limitations of page-based LLCs by transferring a smaller amount of data (block size) between in-package and off-package DRAMs. This, however, demands managing a large amount of tag information. Considering the conventional 64-byte block size, a 512 MB DRAM LLC requires approximately 48 MB of tag storage. Storing the tags in SRAM ensures fast tag lookup but is impractical due to large area and energy requirements [19]. State-of-the-art approaches thus store the tag information alongside cache lines in the LLCs DRAM [11]–[20].

Direct-mapped DRAM LLCs [12], [13] are optimized for LLC hit latency at the expense of a high LLC miss rate. In contrast, associative DRAM LLCs reduce the LLC miss rate but suffer from high hit latency caused by the time required to serially access tags and data. To mitigate this tag serialization effect, previous works [14]–[17] have proposed a small low-latency SRAM tag-cache, which provides fast LLC tag lookup. However, these approaches demand high in-package bandwidth to fetch the tags into the tag-cache after a tag-cache miss. For streaming applications, this results in performance degradation.

The performance and bandwidth utilization of block-based DRAM LLCs can be improved by bypassing the LLC for dead cache lines, i.e., those lines that are not referenced again until eviction. Approaches such as [13], [24] have demonstrated that bypassing indeed alleviates the bandwidth pressure by minimizing the number of unnecessary cache fills. However, even with bypassing, the tag lookups for LLC bypasses incur a large performance overhead and deteriorates the LLC performance, particularly when the number of LLC bypasses is significant.

In this paper, we propose a *tag-data decoupled cache* design for *block-based* DRAM LLCs. Our design reduces conflict misses using a combination of associative caching and LLC bypassing. It retains the benefits of associative LLCs while overcoming their limitations by reducing the tag serialization effect, the number of tag lookups, and the in-package traffic.

The *tag-data decoupled cache* design is based on four key ideas. First, similar to [14], [16]–[18], [20], [24], the decoupled design stores tags in the DRAM LLC. However, unlike all previous approaches, it organizes the LLC into two regions: *tag region* and *data region*. The tag region maintains the metadata of the blocks stored in the data region. The relevant rows of the tag and data regions can be activated and accessed in parallel. The tag region is optimized for low latency, by using smaller mat and row sizes compared to the data region.

Second, the data region is optimized for spatial locality by modifying the mapping of main memory blocks to a cache set. Traditionally, spatial adjacent memory blocks are mapped to different cache sets. In our proposal, multiple spatially-adjacent memory blocks (i.e., *segments*) are mapped to the same cache set. Therefore, the hit/miss information of all blocks belonging to the same segment can be identified by a *single* set access. This optimization reduces the number of set accesses which in turn reduces contention in the data region.

Third, the decoupled design introduces a small *DRAM Absence Table* (DAT) which stores the absence of the recently-accessed LLC-bypassed *super-segments*. A super-segment contains one or more segments which can be configured at design time. This ensures that a future LLC block miss to the same super-segment will be accurately detected by the DAT, thus reducing the number of unnecessary LLC tag lookups.

Finally, we propose an efficient *selective in-DRAM-tag-comparison* policy providing fast LLC tag lookup compared to existing approaches that always perform in-DRAM-tag-comparison [25]. The selective nature of our lookup reduces contention in the tag region.

The major contributions of this paper are:

- A *tag-data decoupled cache* design that mitigates the tag serialization latency by enabling concurrent accesses to the tag and data regions.
- A latency-optimized design for the tag region via an efficient *selective in-DRAM-tag-comparison* policy.
- A *DRAM Absence Table* (DAT) design that provides fast and accurate detection of LLC misses, for blocks that are likely to bypass the LLC in future.
- A novel mapping mechanism that reduces the number of lookups in the tag and the data regions.
- A detailed design space analysis of the *tag-data decoupled cache* design to evaluate the impact of architectural parameters on performance.

Our *decoupled cache* design provides better performance (11.7% BEAR [13], 7.2% LAMOST [16], [20], 11% sector-cache [22], 7.5% TIMBER [15], and 4.7% ACCORD [23]) compared to state-of-the-art designs when an iso-area DRAM LLC is employed.

TABLE 1
Comparisons of state-of-the-art row organizations for a 4 KB row size T_{set}: Number of tag columns in an LLC set, S_{row}: Number of sets in an LLC row,T_{row}: Number of tag columns in an LLC row, A: Associativity

Policy	T_{set}	S_{row}	T_{row}	A
LH-Cache [18], [19]	6	1	6	58
LAMOST [16], [17], [20]	1	8	8	7
ATCache [14]	2	4	8	14

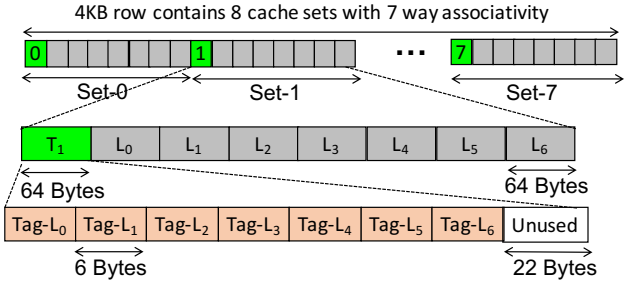

Fig. 1. Row organization of the baseline associative DRAM LLC (Row size = 4 KB)

2 Background

In this paper, we focus on the design of hardware-managed high capacity block-based DRAM LLCs. Block-based LLCs are categorized into direct-mapped [12], [13] and associative [11], [14], [16]–[20] designs. Both approaches store tags and data in the same row of a DRAM bank.

2.1 Associative DRAM LLCs

Previous research studies have proposed many variants of associative DRAM LLCs [14], [16]–[20]. Each DRAM row is divided into S_{row} cache sets and T_{row} tag columns. Every cache set is comprised of T_{set} tag columns and A cache lines. An LLC hit requires access to the T_{set} tag columns before accessing the corresponding cache line.

Table 1 shows the values of S_{row}, T_{set}, T_{row} and A of existing LLC organizations for a 4 KB row size. We consider LAMOST [20], an associative DRAM LLC, as baseline and illustrate the details of its row organization in Fig. 1. Each 4 KB LLC row in the baseline design consists of 8 sets (i.e. $S_{row} = 8$) where every set stores the tags of 7 cache ways (i.e. $A = 7$) in a single tag column (i.e. $T_{set} = 1$). To service an LLC hit, the tag column of the corresponding set is read from the row buffer and checked for a hit/miss. Subsequently, the cache line is accessed from the row buffer whose location is identified by the tag match.

2.2 Baseline LLC with tag-cache and predictor

The baseline LLC aims at reducing the LLC miss rate via associativity. However, this reduction comes at the expense of increased LLC hit latency. To address this problem, recent works employ a small low latency SRAM-based tag-cache that keeps the tag columns of spatially adjacent LLC sets [14]–[17]. Only recently accessed tag columns, not the relevant cache lines, are maintained in the tag-cache. The tag-cache provides fast lookup due to its small size and is accessed before every LLC access. If the tag-cache hits, the relevant tag column from tag-cache is accessed to identify

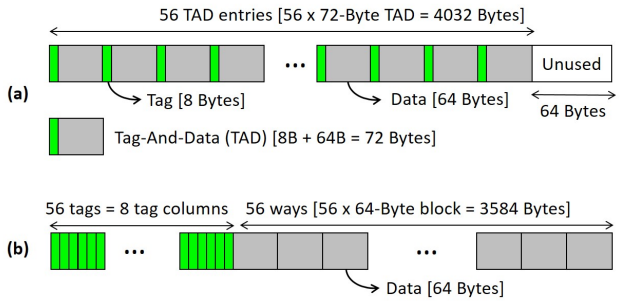

Fig. 2. Row organization of direct-mapped DRAM LLC (a) BEAR [12], [13] (b) TIMBER [15] for (Row size = 4 KB)

an LLC hit/miss. A hit in the tag-cache avoids accessing the tag column from DRAM. As a result, requests that hit in the tag-cache are served much faster than those that miss in the tag-cache. On the contrary, a miss in the tag-cache requires accessing the relevant tag column from the DRAM LLC before accessing the requested cache line (for an LLC hit). Subsequently, the adjacent tag columns (i.e. tags of the adjacent LLC sets) are filled in the tag-cache. The tag-cache exploits the fact that adjacent tag columns will most likely be accessed in the near future.

Similar to [11] and [14], the baseline LLC employs a small MAP-I predictor [12] having a one cycle latency. After a tag-cache miss, the MAP-I predictor is consulted to predict an LLC hit/miss. The off-package DRAM memory is accessed in parallel to the in-package DRAM LLC if MAP-I predicts an LLC miss. Otherwise, the off-package DRAM is accessed only if an LLC miss is identified by the LLC tag. Similar to [26], we bypass the tag-cache for the writeback accesses due to their poor spatial locality.

2.3 Direct-mapped DRAM LLCs

Direct-mapped caches are employed to address the high latency and bandwidth limitations of the associative designs [12], [13], [15]. The direct-mapped BEAR [12], [13] arranges tag and data side by side in a DRAM row to constitute a single *Tag and Data* (TAD) entry, as shown in Fig. 2(a). Compared to associative designs, direct-mapped designs have smaller LLC hit latency because a single access of the TAD entry is required from the row buffer instead of 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 associative designs. The direct-mapped TIMBER groups the tags of 56 cache sets into 8 tag columns [15]. The tags are accessed separately from the data. The TIMBER design for a 4 KB row size is depicted in Fig. 2(b) and it is is backed up by a tag-cache that stores the tags of spatially adjacent cache sets.

2.4 Sector DRAM LLC

In sector DRAM LLC [22], the basic unit of cache allocation is referred to as sector which is comprised of multiple blocks. Fig. 3(a) shows a sector organization with 8 blocks per sector. Each sector is associated with a tag (Sector-Tag) to determine whether an entry is reserved for a particular sector (i.e., sector hit) or not (sector miss). Each block b_i of a sector is provided with a presence bit P_i and thus only some

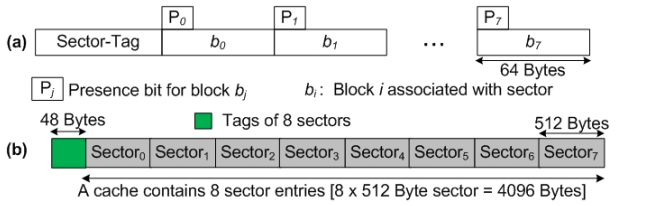

Fig. 3. (a) Sector organization with 8 blocks per sector (b) Row organization of sector DRAM LLC (Row size = 4 KB and sector size = 512 Bytes)

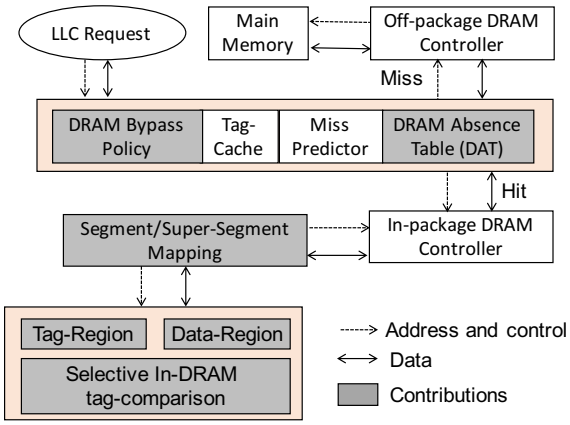

Fig. 4. High level view of our *Decoupled Cache* Design.

blocks of a sector need to be present in the LLC. Following a sector hit, when a block b_i of a particular sector S misses the LLC (i.e., block miss), then P_i of S is set. If no entry for S exists in the LLC (i.e., sector miss), a new entry is reserved for S and only the presence bit of b_i (i.e. P_i) is set.

Sector cache fetches data at block-level (i.e. 64 bytes) in order to mitigate the excessive prefetching problem of page-based LLCs [5], [8]–[10]. The work in [22] stores the sector tags in the DRAM row along with the data and uses a tag cache to hold the tags of recently accessed sectors. Fig. 3(b) shows the row organization of a sector based DRAM LLC assuming that the DRAM row size is 4 KB and the sector size is 512 bytes. The sector organization in the figure corresponds to an 8-way associative cache where the tags of 8 sectors are stored in a separate tag column (highlighted in green).

3 The Decoupled Cache Design

A high level overview of the proposed *decoupled cache* design is shown in Fig. 4. Similar to [11], [12], [14], we use the MAP-I predictor from [12] and the tag-cache from [14], [16], [17]. Proposed changes to realize our decoupled design are highlighted in grey color which are explained in the following subsections.

3.1 Decoupled tag region and data region

Fig. 5 shows the organization of the tag and data regions in our *decoupled cache* design. The data region consists of $N \times 4$ data banks while the tag region consists of N tag banks, where N refers to the number of bank groups. Each bank, in both regions, consists of R logical rows. Each row in the data region has 8 sets (i.e. $S_{row} = 8$). Each set is made up of A cache lines. Each row of a tag bank is partitioned

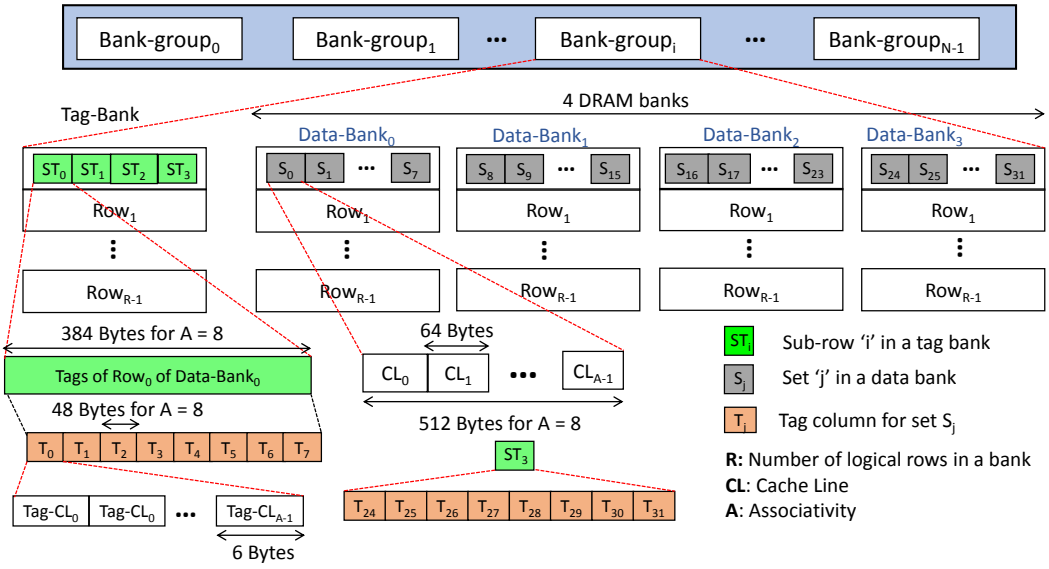

Fig. 5. Logical organization of the *decoupled cache* design

into 4 sub-rows where each sub-row stores the tags of its corresponding rows in the data banks (from the same bank group). The decoupled design provides the following advantages compared to existing block-based associative DRAM LLCs in general and the baseline LLC in specific.

Concurrent tag-data accesses: The requested rows of the tag and the data region are activated in parallel after a tag-cache miss. In contrast, the baseline architecture suffers from increased tag serialization latency because it accesses the tag column before accessing the cache line for a tag-cache miss (cf. Fig. 10-b).

Latency optimized tag region: The tag region in the decoupled design is accessed more frequently compared to the data region. We exploit this insight and optimize it for latency by employing smaller row and mat sizes (cf. Table 2). Although smaller mat sizes consumes more area due to increase in the number of sense amplifiers, the overall impact on the area utilization is insignificant due to the small size of the tag region.

Latency optimization for data region: A victim row in the row buffer of the data region is clean most of the time. As a result, it does not incur t_{WR} latency for a clean victim row. This penalty is only incurred for the tag region which updates the tag columns after their eviction from the tag-cache. However, the negative impact of t_{WR} in the tag region is less due to its smaller access latency. In contrast, existing tag-data coupled LLC designs always incur a latency penalty t_{WR} even for clean cache lines.

Efficient utilization of the storage space: The decoupled design eliminates the wastage of storage space in existing block-based DRAM LLC designs. For instance, each 64-byte tag column in the baseline LLC stores the tags of a cache set as depicted in Fig. 1. Each cache line requires 6 bytes for the tag information. Therefore, the 7 cache lines of a set require $7 \times 6 = 42$ bytes for their tag entries leaving 22 bytes per tag column unused (cf. Fig. 1). For a 512 MB LLC, a total of 22 MB is left unused (22 bytes / tag column $\times$ 8 tag columns / row $\times$ 4096 rows / bank $\times$ 32 banks). Likewise, other associative [18] and direct-mapped [12], [13] designs (cf. Fig. 2) waste considerable amount of storage space. Decoupling the tags from the data makes it possible to use a smaller column size for the tag banks, considerably reducing the space wastage. As an example, using a tag column size of 42 bytes instead of 64 bytes will eliminate this wastage.

3.1.1 Definitions and Design Space

This section introduces the terminology used in the following sections. Further, it elaborates on the important parameters of the *decoupled cache* design.

- **Block:** A block is a group of contiguous bytes in main memory.
- **Segment:** A segment consists of K spatial adjacent blocks.
- **Super-segment:** A super-segment consists of T spatial adjacent segments.
- **Super-set:** A super-set consists of T spatial adjacent sets.

It is worth mentioning that T tag column accesses are required to identify the LLC hit/miss information of all blocks belonging to a super-segment. For elaboration, we make the following assumptions about the design space.

- The block size is 64 bytes.
- The segment size is 256 bytes, corresponding to 4 (i.e. $K = 4$) spatial adjacent blocks.
- The super-segment size is 1 KB, corresponding to 4 (i.e. $T = 4$) spatial adjacent segments and 16 spatial adjacent blocks.
- The row size in the data region is 4 KB and it is comprised of eight 512-byte sets.
- Each 512-byte set has 8-way associativity (i.e. $A = 8$).

For the purpose of explanation, a constant value of K, T, and A is considered. However, the concepts proposed in this paper can be applied to other settings as well. Section 4.5 provides a detailed design space exploration by varying the values of K and T. Similarly, the *decoupled cache* design is

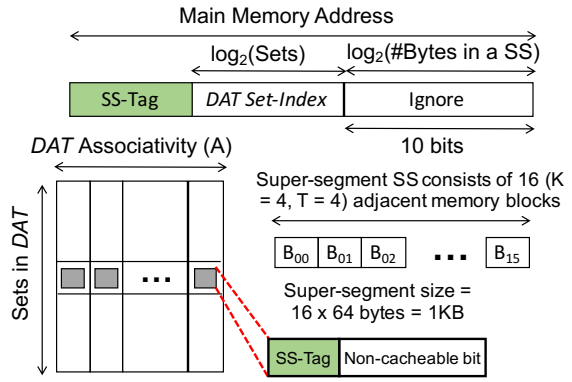

Fig. 6. DRAM Absence Table (DAT) organization

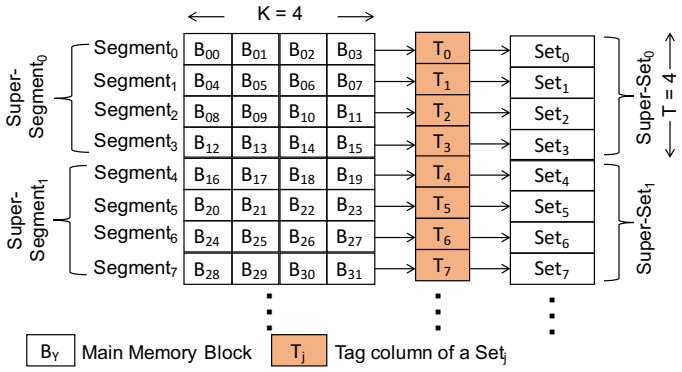

Fig. 7. Segment mapping to the tag columns. K: Segment size

evaluated for 8-way (Section 4.4) and 6-way (Section 4.7) caches as well.

3.2 DRAM Absence Table (DAT)

The decoupled design uses DAT to quickly identify the LLC absence information. The DAT is organized as a set-associative cache which stores the absence information of super-segments as illustrated in Fig. 6. It maintains a single bit to specify non-cacheable super-segments. The value of non-cacheable bit is 1 if all blocks of a super-segment are absent in the LLC. An in-DRAM tag lookup is avoided for an LLC block miss that belongs to a non-cacheable super-segment. Our *decoupled cache* design directly forwards this request to the off-package DRAM and is never filled in the in-package DRAM LLC. Leveraging the non-cacheable bit of the super-segment, the number of in-DRAM tag lookups for blocks that belong to the same super-segment is reduced. The DRAM Access Table (DAT) provides coarse-granularity miss information at super-segment level that requires access to T tag columns. It is worth to mention that super-segment is either equal to segment (i.e., T = 1) or greater than segment (i.e., T >1). A DAT tracks the hit/miss information of $sets\_in\_DAT$ x $ways\_in\_DAT$ recently accessed super-segments that bypasses the LLC. The total memory covered by DAT is $sets\_in\_DAT$ x $ways\_in\_DAT$ x super-segment-size. The super-segment-size in bytes is equal to K x T x 64 assuming a 64-byte cache line size. This implies that the size of a super-segment is configurable at design time by choosing a suitable value of K and T. For $sets\_in\_DAT$ = 64, $ways\_in\_DAT$ = 4, K = 4 and T = 4, the memory covered by DAT is 256 KB. Note that only a limited number of recently bypassed super-segments are maintained in the small 1 KB DAT (cf. Section 3.6).

3.3 Segment Mapping

The in-package bandwidth can be effectively utilized for useful data transfer if tag region accesses are reduced. This can be achieved by reducing the number of tag-cache/DAT misses. To this end, we modify the mapping of segments to the tag columns of an LLC set. Fig. 7 shows this block mapping for $K = 4$. For $K = 1$, this mapping is equivalent to the existing block-based DRAM LLCs [12], [13], [18]–[20]. With the new mapping, a single tag column access is sufficient to identify the LLC hit/miss status of all K blocks of the same segment. For instance, as depicted in Fig. 7, only access to T_2 is required to determine the hit or miss for all 4 blocks (i.e., B_{08} to B_{11}) belonging to $Segment_2$.

Varying the segment size (i.e. K) provides an interesting trade-off between the tag-cache/DAT hit rate and the LLC miss rate. A larger segment size implies a higher tag-cache/DAT hit rate compared to a smaller segment size. A large segment size leverages the spatial locality by mapping more blocks to the same tag column or set. However, restricting all K blocks of the same segment to reside on the same set may negatively affect the LLC miss rate. The smallest segment size (i.e. $K = 1$) eliminates within-set contention because spatially adjacent blocks are directed to different LLC sets. This comes at the cost of reduced tag-cache/DAT hit rate. The detailed trade-off analysis is provided in section 4.5.1.

3.4 DRAM Bypass Policy

A *miss fill* refers to an operation that inserts a new block in the cache after a cache miss [13]. If the newly inserted block is not re-referenced before its eviction, it is referred to as dead block. Inserting a dead block after a cache miss is referred to as *useless miss fill*. Often, these useless miss fills replace useful resident cache lines, i.e. those potentially to be accessed in the near future. An efficient LLC bypass scheme can improve the LLC performance by reducing the number of useless miss fills [13], [24]. However, the LLC-bypassed dead blocks still require in-DRAM tag lookups for correction which unnecessarily consume bandwidth and cause bank conflicts.

Based on their cache reuse behavior, applications are classified into two categories, streaming applications and cache-friendly applications [27]–[29]. Streaming applications generate significant number of useless miss fills. Therefore, it is wise to bypass the DRAM LLC for such applications. Contrarily, majority of blocks belonging to cache-friendly applications exhibit high temporal locality and are frequently reused.

For LLC bypassing, set dueling [30] is used to dynamically select between two insertion rates (1 and $\frac{1}{32}$). For each core, two sampling monitors are used to track the miss statistics of $\frac{NS_s}{64}$ super-sets. NS_s refers to the total number of super-sets in the DRAM LLC. The remaining super-sets are referred to as follower super-sets. The sampling monitors are highlighted with grey and blue colors in

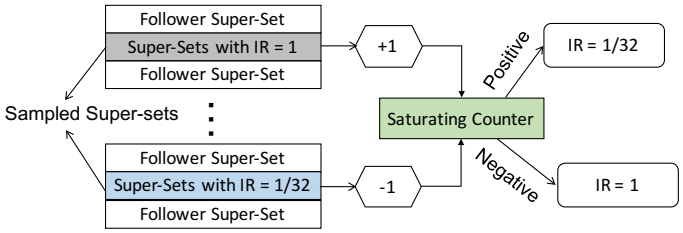

Fig. 8. DRAM Bypass Policy

Fig. 8. The two sampling monitors employ insertion rates of 1 and $\frac{1}{32}$ respectively. A 10-bit saturating counter (green box) is used to keep track of the misses in each sampling monitor. The counter value is incremented (or decremented) on a miss incurred in the grey (or blue) sampling monitor respectively. The most significant bit (MSB) of the counter decides the insertion rate for the follower super-sets. The saturating counter calculates which of the two insertion rates (i.e. 1 or $\frac{1}{32}$) will reduce the DRAM LLC miss rate. Similar to [13], [24], our DRAM bypass policy is based on the tenet of selecting low insertion rate (e.g. $\frac{1}{32}$) for streaming applications and a high insertion rate (e.g. 1) for cache-friendly applications. However, in contrast to the previous policies, this paper proposes DRAM bypassing at the super-segment level instead of the block level. Note that a super-segment comprises 16 (i.e. $K = 4$ and $T = 4$) spatial adjacent blocks (i.e. b_0 to b_{15}) and it is mapped to one of the super-sets (see Fig. 7).

The following example is considered to illustrate our proposed bypass, tag-cache fill and DAT fill policies. Assume that a block b_k ($0 < k < 15$) from a particular super-segment SS_i is currently requested with a miss in both the tag-cache and the DAT. If any block of the same super-segment SS_i is present in the LLC then the *decoupled cache* performs the following. (a) b_k is inserted into the LLC (b) all tag columns belonging to the super-segment SS_i are filled into the tag-cache. However, if all blocks of the super-segment SS_i are absent in the LLC then the bypass decision depends on the outcome of the DRAM bypass policy. If the bypass policy decides to bypass the super-segment SS_i, then all future accesses to other blocks of the same super-segment SS_i are also bypassed. In that case, the non-cacheable bit in DAT is set for the super-segment SS_i. If the outcome of the bypass policy is to insert the super-segment SS_i, then b_k is inserted into the LLC and the T tag columns are fetched into the tag-cache.

3.5 DRAM Control Flow

Fig. 9 shows the control flow of the decoupled design to serve an LLC request. For every LLC request, the tag-cache and the DAT are accessed to identify an LLC hit/miss. A tag-cache hit precisely identifies an LLC hit/miss and does not require an in-DRAM tag lookup. If a request hits in the DAT, i.e. the non-cacheable bit of the super-segment is 1, it is directly forwarded to the off-chip memory. In this case, in-DRAM tag lookup is avoided because the requested block is guaranteed to be absent in the LLC. A miss in both the tag-cache and the DAT does not provide any clue about the presence/absence of the block, thus requires an in-DRAM

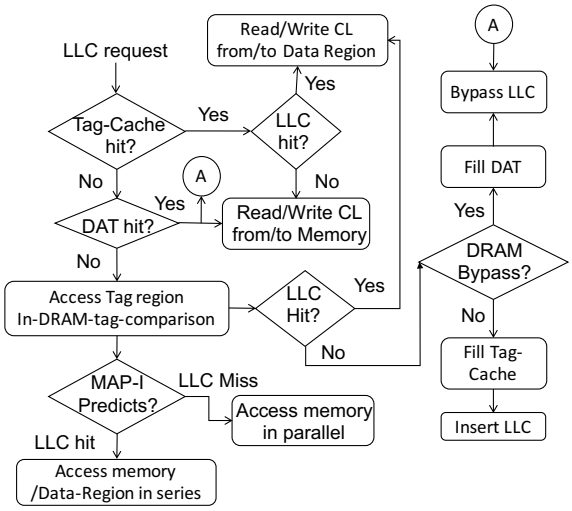

Fig. 9. DRAM LLC control flow with DAT and tag-cache

tag lookup to identify an LLC hit/miss. This needs access to $T = 4$ tag columns of the tag region.

Managing bypassed blocks: To illustrate the significance of the DAT in our architecture, let us consider a sequence of 9 block accesses b_7, b_3, b_4, b_6, b_8, b_0, b_{12}, b_{14}, b_{15} belonging to the super-segment SS_i. It is assumed that SS_i currently misses the DAT, the tag-cache and the LLC. It is further assumed that the DRAM bypass policy decides to bypass SS_i. For the existing LLC bypass policy [13], the above sequence incurs nine in-DRAM tag lookups. In contrast, our DRAM bypass policy requires a single in-DRAM tag lookp as explained below. The outcome of the DAT is a miss after accessing block b_7. Since the bypass policy decides to bypass SS_i, block b_7 bypasses the LLC. The non-cacheable bit of SS_i is set in the DAT because all blocks of the SS_i are currently absent in the LLC. For all subsequent accesses to other blocks of the SS_i (i.e. b_3, b_4, b_6, b_8, b_0, b_{12}, b_{14}, b_{15}), the DAT precisely determines an LLC miss without incurring any in-DRAM tag lookup. In this case, the request is directly forwarded to the off-chip memory.

Managing non-bypassed blocks: To illustrate the benefits of the tag-cache in our proposal, it is assumed that the blocks b_6, b_8, b_{12}, b_{14}, and b_{15} of the above sequence are currently present in the LLC with a miss in both the DAT and the tag-cache. In this scenario, an access to the block b_7 of the super-segment SS_i results in an LLC miss. However, some blocks of the super-segment SS_i are currently present in the LLC which are identified by the 4 tag columns of the tag region. The *decoupled cache* first performs an in-DRAM tag lookup to identify hit/miss for the block b_7. Subsequently, all the tag columns are filled in the tag-cache in burst mode. The remaining eight blocks (i.e. b_3, b_4, b_6, b_8, b_0, b_{12}, b_{14}, b_{15}) then hit in the tag-cache, thus avoiding in-DRAM tag lookups.

Our proposal provides quick identification of the LLC hit/miss because:

- Cache-friendly applications significantly benefit from high tag-cache hit rate due to reduced number of tag region accesses.
- The DAT provides fast LLC miss detection for majority

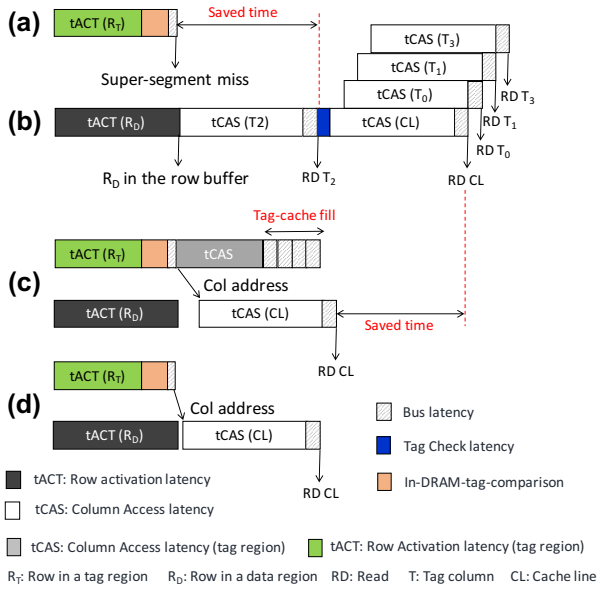

Fig. 10. (a) Super-segment miss detection in the tag region for the bypassed super-segment (b) Hit/miss latency in the baseline combined tag-data region (c) Hit latency of the decoupled design where tag and data regions are accessed concurrently along with tag-cache fill (d) In-DRAM tag comparison in [25] without a tag-cache.

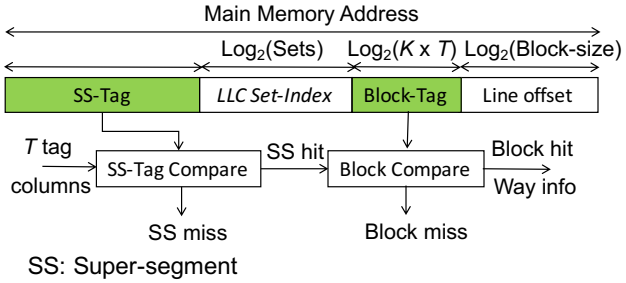

Fig. 11. Selective in-DRAM-tag-comparison

of accesses from streaming applications avoiding access to both the tag region and the data region.

- The MAP-I predictor quickly forwards read requests to the off-chip memory if it predicts an LLC miss.

3.5.1 Parallel tag-data access

Following the tag-cache and the DAT miss, the relevant rows of the tag and data regions are accessed in parallel; this happens only if the DRAM bypass policy decides to insert the requested block in the LLC or the MAP-I predicts an LLC hit. However, if the DRAM bypass policy decides to bypass the requested block, then only the relevant row of the tag region is accessed.

Let us consider the worst case scenario where the relevant rows of both the tag and the data region are absent in the row buffers. It is assumed that the requested block is present in the LLC and its tag information is stored in one of the tag column residing in the relevant row of the tag region. Existing associative LLC designs read the tag column and the cache line in a sequential fashion as shown in Fig. 10(b). On the contrary, the decoupled design accesses the relevant rows of the tag and the data regions in parallel and overlaps their latencies as depicted in Fig. 10(c).

3.5.2 Selective in-DRAM-tag-comparison

State-of-the-art LLCs always perform an in-DRAM-tag-comparison for each requested block (cf. Fig. 10d) which incurs high latency due to an increased number accesses to the DRAM banks storing tags [25]. To address this problem, an in-DRAM-tag-comparison is performed only after the tag-cache and the DAT miss. The in-DRAM-tag-comparison needs to be performed on T tag columns (i.e. $T = 4$). This comparison is divided into two parts, namely super-segment tag and block tag comparisons, as depicted in Fig. 11. As illustrated, the *SS-Tag* field of the memory address identifies a particular super-segment and the *Block-Tag* field identifies the block within that super-segment. A super-segment miss implies that all blocks of that super-segment do not reside in the LLC.

Our decoupled design only requires a single super-segment tag comparison to update the non-cacheable bit in the DAT for the bypassed super-segment as illustrated in Fig. 10(a). Subsequent miss detection of blocks belonging to the same super-segment will be identified by the low-latency DAT. Note that there is no need to transfer T tag columns via in-package channel to update an entry in the DAT (cf. Fig. 10a). In contrast, these transfers are required to update an entry in the tag-cache (Fig. 10c). This selective in-DRAM-tag-comparison significantly reduces contention in the tag region compared to [25]. Furthermore, the super-segment miss detection obviates the need for the CAS command in the tag region which is required to fetch the tags in the tag-cache.

Similarly, a single in-DRAM-tag-comparison is required to identify the hit/miss information for the blocks belonging to a non-bypassed super-segment. It is worth to mention that the tag-cache fill (tag region) is performed concurrently with the cache line access (data region). As highlighted in Fig. 10(c), the CAS (column access latency) of the tag and data regions are overlapped.

3.6 Overhead Analysis

The tag-cache stores the tag information of the recently accessed super-sets. Every super-set comprises T tag columns where every tag column is 48 bytes ($6 \times 8 = 48$ bytes) for an 8-way associative DRAM LLC, assuming 6 bytes for the LLC tag (see Fig. 5). For $T = 4$, the storage requirement of a single entry in the tag-cache is 196 bytes which comprises (a) 192 bytes for storing four tag columns ($48 \times 4 = 192$) (b) 22 bits for the super-segment tag (c) one valid bit to indicate whether the super-segment is valid or not (d) one dirty bit to indicate whether the tag columns need to be updated in the tag region or not. A 4-way associative tag-cache is employed with 64 sets. Therefore, the total storage requirement of the tag-cache amounts to 49 KB (196 bytes/way $\times$ 4 ways/set $\times$ 64 sets).

The DAT stores the absence information of the recently bypassed super-segments. Every DAT entry needs 22 bits for the super-segment tag, one bit each for the valid and non-cacheable fields. Therefore, the size of each DAT entry is 4 bytes. We employ a 4-way associative DAT with 64 sets. Thus, the DAT requires a trivial additional overhead of 1 KB (4 bytes/way $\times$ 4 ways/set $\times$ 64 sets = 1 KB) which is negligible compared to the storage overhead of the tag-cache. It

TABLE 2
Comparisons of configuration parameters for the tag and data regions employing 512 MB DRAM LLC

Parameters	Tag Region	Data Region
# banks	8	32
# Rows in a bank	4096	4096
# Mats in a bank	128	128
Mat Size	512 x 768 cells	1024 x 1024 cells
Row size	1536 Bytes	4096 Bytes
Total size	48 MB	512 MB
Total area	26.6 mm^2	107.5 mm^2
t_{RAS}-t_{RCD}-t_{RP}-t_{CAS}-t_{WR}	$17 - 4.25 - 4.25 - 4.25 - 4.25(ns)$	$24 - 6 - 6 - 6 - 6(ns)$

TABLE 3
System Parameters Details

Core	3.2 GHz
Shared SRAM	8 MB, 8-way associativity, 5 nsec latency
Shared DRAM LLC	4 channels, 4 KB row buffer, 512 MB, 32 banks, 128-bit channel width, 1.6 GHz bus frequency and t_{RAS}-t_{RCD}-t_{RP}-t_{CAS}-t_{WR} = $24 - 6 - 6 - 6 - 6(ns)$
tag-cache	49 KB, 0.625 nsec latency [14], [16], [17]
Miss Predictor	Map-I [12], 256 entries
Main Memory (DRAM)	2 channels, 16 KB row buffer, 64-bit channel width, 800 MHz bus frequency and t_{RAS}-t_{RCD}-t_{RP}-t_{CAS}-t_{WR} = $36 - 9 - 9 - 9 - 9(ns)$

is pertinent to mention that the only additional overhead of the *decoupled cache* is the DAT overhead (1 KB) because tag-cache is already employed in state-of-the-art approaches. The area overhead analysis is provided in Section 4.7.

4 Evaluation

This section presents a comprehensive evaluation of the *decoupled cache* by comparing it with state-of-the-art associative [20], [25], [31] and direct-mapped [12] designs.

4.1 Experimental Setup

Table 3 presents the details of the system and memory parameters used in our evaluations. We use DRAMSpec [32] to estimate the area and latency values of the tag and the data regions. These values are presented in Table 2 and Table 3 respectively. The cycle-accurate NVMain simulator [33] is extended to model our *decoupled cache* design [34]. Further, the simulator trace-reader is modified to support multi-program workloads. All experiments are performed by forwarding transactions to the simulator from eight different trace files which mimics an 8-core system. All simulations are carried out in two switchable modes, namely non-timing and timing mode. In the non-timing mode, 3 billion instructions are simulated from each application to warm-up shared-SRAM and shared-DRAM caches. Subsequently, timing simulation is performed for 600 million instructions per application to gather performance statistics.

In particular, the following evaluated configurations are analyzed in detail:

1) **DFC-INCLUSIVE:** The recently proposed inclusive cache hierarchy that fuses the tags of shared DRAM cache with the tags of shared SRAM cache [31]. This configuration benefits from the large cache line size. A 256-byte cache line is considered as it outperforms configurations with other cache line sizes. A decoupled tag-data design is assumed for this configuration with 8-way associativity.
2) **BEAR:** Direct-mapped cache (Section 2.3) with bypass support as in [24] .
3) **BASE-T$:** Baseline LLC [20] as described in Section 2.2 with block-based bypass support as in [24]. We use a segment size of 64 bytes (i.e. $K = 1$) and prefetch 4 tag columns in the tag-cache [14], [16].
4) **IDTC-A8:** Existing 8-way LLC with $K = 1$ and bypass support as in [24]. This configuration always performs in-DRAM-tag-comparison in the tag region similar to [25]. A decoupled tag-data design is assumed for this configuration.
5) **DEC-A8:** Our 8-way associative *decoupled cache* as described in Section 3.1 with $T = 4$ and $K = 4$ (Section 3.3), super-segment level bypassing (Section 3.4), selective in-DRAM-tag-comparison (Section 3.5), and tag-cache/DAT (Section 3.2) support.

For all listed configurations, the following assumptions are made:

- The size of the DRAM LLC is 512 MB and the row buffer size is 4 KB. The DRAM configuration parameters are presented in Table 2.
- The MAP-I predictor [12] is used to quickly predict an LLC hit/miss.
- First ready first come first serve (FR-FCFS) scheduling [35] is employed while request starvation is avoided with a set threshold.

TABLE 4
SPEC2006 Workloads—LLC intensive application (shared-SRAM $MPKI > 12$) are highlighted in bold

Mix1	*zeusmp*, *bzip2(2)*, *cactusADM*, *sphinx3*, *xalancmbk*, *wrf*, *GemsFDTD*
Mix2	**soplex**, *wrf(2)*, *sphinx3*, **libquantum**, *bzip2*, *leslie3d(2)*
Mix3	*bzip2*, *cactusADM(2)*, **mcf**, *xalancmbk*, **milc(2)**, *GemsFDTD*
Mix4	*Zeusmp*, *xalancmbk*, *sphinx3(2)*, **milc**, *leslied3d*, **bwaves(2)**
Mix5	*GemsFDTD*, *leslie3d*, **omnetpp**, *xalancmbk*, *bzip2*, **lbm**, **libquantum**, **mcf**
Mix6	*cactusADM*, *Zeusmp*, **mcf**, **soplex**, **lbm(2)**, **milc**, *sphinx3*
Mix7	**omnetpp(2)**, **bwaves**, **milc(2)**, **gcc**, *zeusmp*, *wrf*
Mix8	**soplex(2)**, **libquantum(2)**, **lbm(2)**, **mcf(2)**

4.2 Application Classification

We use SPEC2006 benchmarks traces from [36]. The applications are classified into two categories namely LLC intensive and LLC non-intensive based on the shared-SRAM misses per thousand instructions (MPKI). The aforementioned configurations are evaluated by executing 8 workloads from the SPEC2006 benchmarks [37]. The workloads are constructed such that they represent a mix of LLC intensive and non-intensive applications to ensure a comprehensive evaluation. The details of the evaluated workloads are given in Table 4 where the LLC intensive applications (having shared-SRAM MPKI > 12) are highlighted in bold.

TABLE 5
Comparisons of different configurations in terms of associativity (A), segment size (K), number of tag columns in a row (T_{row}), and number of spatial adjacent blocks mapped to the same row (Num_SAB_Row). Row size is assumed to be 4 KB.

Configuration	A	K	T_{row}	Num_SAB_Row ($K \times T_{row}$)
DFC-INCLUSIVE	8	4	2	8
BEAR & TIMBER & ACCORD (Fig. 2)	1	1	56	56
BASE-T$ (Fig. 1)	7	1	8	8
IDTC-A8$	8	1	8	8
DEC-A8 (Fig. 5)	8	4	8	32
SectorCache (Fig. 3)	8	1	1	8

4.3 Discussion

The system performance is highly influenced by LLC miss rate (lower is better) and LLC read hit latency (lower is better). However, these metrics are affected in a conflicting manner by two DRAM LLC parameters: associativity and Num_SAB_Row. The Num_SAB_Row is defined as the number of spatial adjacent blocks that are mapped to the same DRAM row. Increasing the value of Num_SAB_Row reduces the LLC read hit latency by improving the LLC row buffer hit rate (higher is better) and the tag-cache hit rate (higher is better). A high row buffer hit rate implies that most of the LLC accesses will be directly accessed from the row buffer, thereby avoiding the costly DRAM array accesses. Likewise, a high tag-cache/DAT hit rate indicates that most tag lookups will not require any access to the DRAM LLC and will be serviced directly by the tag-cache/DAT. A higher Num_SAB_Row exploits the spatial locality by placing many consecutive spatial adjacent blocks to the same row buffer. However, increasing Num_SAB_Row exacerbates the LLC miss rate due to an increased within-row contention among spatial adjacent blocks. Table 5 shows the values of associativity and Num_SAB_Row for all evaluated configurations.

4.4 Evaluation of DEC-A8

The evaluation results show that our architecture does improve LLC miss rate and LLC read hit latency at the same time. This improvement comes from high associativity (i.e. $A = 8$), reduced cache contention, high tag-cache and row buffer hit rates, parallel tag-data accesses, and selective in-DRAM-tag-comparison. A detailed qualitative and quantitative comparison with different configurations is presented in the following.

4.4.1 Comparison with DFC-INCLUSIVE [31]

This section analyzes the impact of segment mapping, small cache line size and LLC bypassing by comparing our DEC-A8 and DFC-INCLUSIVE (without bypass support and using larger cache line size) configurations. On average, DEC-A8 outperforms DFC-INCLUSIVE by 6.1% as depicted in Fig. 12 due to the following reasons.

Impact of segment mapping: For the same 8-way associativity, the Num_SAB_Row parameter for DFC-INCLUSIVE is 8 while that of DEC-A8 is 32 (cf. Table 5). As a result, our DEC-A8 experiences a higher row buffer hit rate (41%) compared to DFC-INCLUSIVE (32.1%) as shown in Fig. 15.

Impact of the cache line size: A larger cache line size exploits the spatial locality by fetching multiple blocks at once while only one block is requested. The use of a larger cache line size (i.e. 256 bytes) in DFC-INCLUSIVE reduces the LLC miss rate from 20.5% to 12.5% compared our DEC-A8 (cache line size is 64-bytes). However, this miss rate reduction comes at the cost of 17.1% increase in the off-package latency (cf. Fig. 15) because multiple 64-byte blocks must be transferred through a limited size memory channel. This exacerbation in the off-package latency is primarily caused by long queuing delay due to tansferring large data through limited off-package bandwidth.

Impact of LLC bypassing: Another drawback of DFC-INCLUSIVE is that it always fetches and inserts 256-byte data into the LLC. This leads to inefficient bandwidth (both in-package and off-package) and LLC storage utilization for applications with low spatial locality. This deterioration occurs because many pre-fetched dead blocks are not reused later. In contrast, DEC-A8 bypasses the LLC for majority of dead blocks that effectively utilize in-package bandwidth.

By using smaller cache line size and avoiding useless prefetches, our DEC-A8 reduces the off-package latency by 17.1% via reduced bandwidth pressure. The LLC read hit latency is reduced by 12.3% that is achieved via a higher row buffer hit rate and LLC bypassing. The performance benefits of DEC-A8 is not uniform across all workloads. For instance, DFC-INCLUSIVE exhibits similar performance to DEC-A8 for Mix1 with reduced miss rate (cf. Fig. 13). This workload under-utilizes in-package and off-package bandwidth due to a lower miss rate and favours a large cache line size. However, DFC-INCLUSIVE performs significantly worst (i.e., Mix5 to Mix8) when the miss rate is higher due to majority of dead block insertions which exacerbates both in-package and off-package contention.

4.4.2 Comparison with BASE-T$ [20], [24]

This section compares two associative bypass-supported configurations. These include our DEC-A8 ($Num\_SAB\_Row = 32$ and $A = 8$) and BASE-T$ ($Num\_SAB\_Row = 8$ and $A = 7$) configurations. As mentioned before, increasing the Num_SAB_Row parameter increases the LLC row buffer hit rate and tag-cache hit rate. The average row buffer and tag-cache hit rates of our DEC-A8 (i.e. 41% and 71.9% respectively) are higher than that of the BASE-T$ (i.e. 28.6% and 41.2% respectively) as demonstrated in Fig. 15 and Fig. 14 respectively. On the downside, increasing the Num_SAB_Row parameter increases the LLC miss rate due to increased within-row contention. However, DEC-A8 alleviates this within-row contention via high associativity (i.e. $A = 8$) compared to BASE-T$ (i.e. $A = 7$) as depicted in Table 5. Therefore, our DEC-A8 notices almost similar LLC miss rate compared to BASE-T$.

As depicted in Fig. 12, our DEC-A8 outperforms BASE-T$ by 11.6% due to significant reduction in the LLC read hit latency (i.e. 27.2%). Following are the six major reasons for this latency reduction. First, most tag lookups are satisfied in the low latency tag-cache due to a high tag-cache hit rate. The tag-cache hit rate is improved by using a larger segment size. In addition, the bypass supported *decoupled cache* design directs requests from streaming applications to the

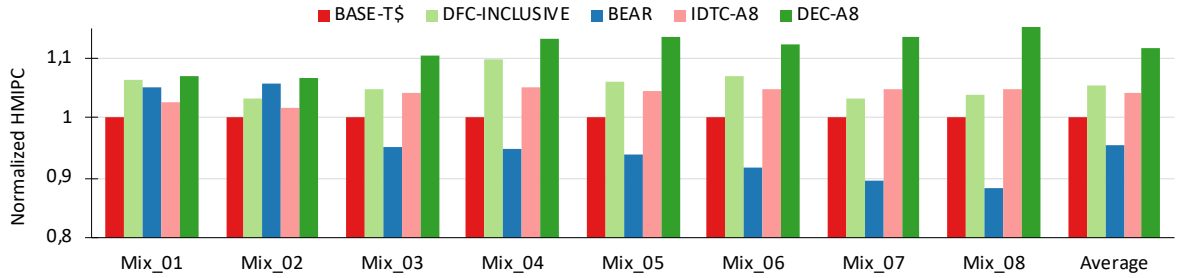

Fig. 12. Normalized Harmonic Mean Instructions per Cycle (HMIPC) results

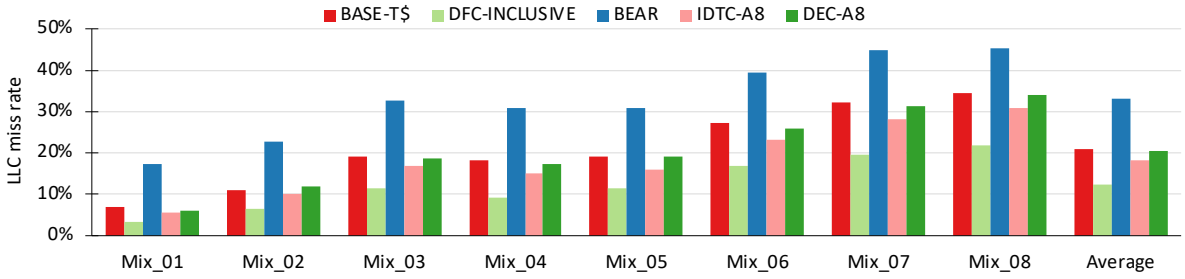

Fig. 13. LLC miss rate results

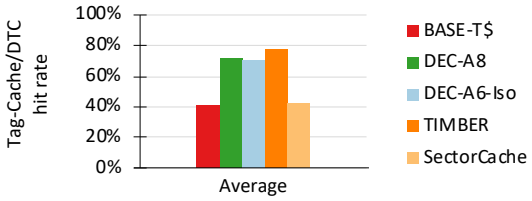

Fig. 14. Tag-cache/DAT hit rate results (DAT is not employed in BASE-T$, TIMBER and SectorCache)

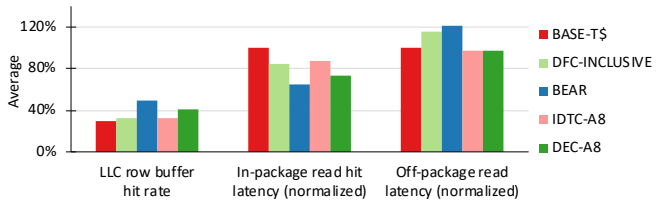

Fig. 15. LLC row buffer hit rate, hit latency and off-package read latency. All results are averaged across all workloads

DAT. Consequently, this reduces pressure on the tag-cache and further improves its hit rate. Second, due to the higher row buffer hit rate of our DEC-A8, more LLC accesses hit in the row buffer. Third, our DEC-A8 grants parallel access to both the tag and data regions which mitigates the tag serialization latency compared to BASE-T$ (cf. Fig. 10c).

Fourth, bypassing requests in BASE-T$ require tag lookups in the high latency combined tag-data region. In contrast, these tag looks are serviced by the latency optimized tag region in our DEC-A8. Fifth, the selective in-DRAM-tag-comparison requires minimal in-package bandwidth to update an entry in the DAT (cf. Fig. 10a). In contrast, BASE-T$ requires many cycles to transfer T tag blocks in addition to column access latency (Fig. 10b). This additional latency overhead is required to update an entry in the tag-cache. Finally, our DEC-A8 saves t_{WR} latency for the data region in the scenario when the corresponding row buffer of the data region is not modified. In contrast, BASE-T$ incurs t_{WR} penalty upon every row buffer eviction because the tags and data are stored in the same LLC row. The modified tag information needs to be updated in the DRAM array upon every row buffer eviction which obligates the extra t_{WR} latency even if cache lines are not modified.

4.4.3 *Comparison with IDTC-A8 [25]*

This section quantitatively and qualitatively analyzes the performance impact of selective in-DRAM-tag-comparion along with tag-cache/DAT support. In contrast to our DEC-A8, IDTC-A8 [25] employs a smaller segment size (i.e., $K = 1$) without a tag-cache/DAT support. Compared to IDTC-A8, the performance improvement of our DEC-A8 translates to 7.5% due to following three reasons. First, IDTC-A8 suffers from increased contention in the tag region as it always necessitates in-DRAM-tag-comparison for all requested blocks of a particular super-segment. This implies that the number of in-DRAM-tag-comparisons is equal to the number of block requests from the same super-segment. In contrast, our DEC-A8 only requires a single in-DRAM-tag-comparison. Second, the subsequent tag comparison of the remaining requested blocks of the same super-segment will be carried out in the low-latency tag-cache or DAT. This further reduces the contention in the tag region. Finally, our DEC-A8 provides higher row buffer rates compared to IDTC-A8 due a high value of Num_SAB_Row parameter.

4.5 Design Space Exploration

This section provides a detailed design space exploration, of the *decoupled cache*, by varying the segment and super-

TABLE 6
Impact of the segment size on parameter Num_SAB_Row

Segment Size (K)	$Num\_SAB\_Row = K \times T_{row}$
1	8
2	16
4	32
8	64

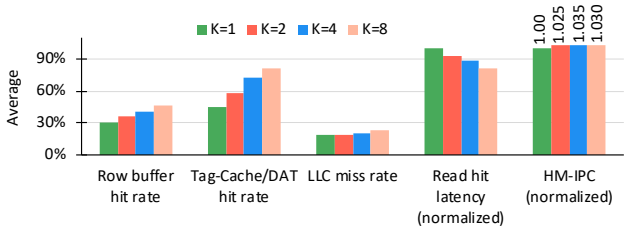

Fig. 16. Impact of the varying segment sizes on various performance metrics ($T = 4$)

segment sizes and observing their impact on various metrics and parameters. For the rest of this section, the LLC row buffer hit rate refers to the row buffer hit rate of the data region.

4.5.1 Varying the segment size

Fig. 16 highlights the impact of the segment size on the LLC row buffer hit rate, tag-cache/DAT hit rate, LLC miss rate, LLC read hit latency and the overall LLC performance. For brevity, the segment size is varied from 64 (i.e. $K = 1$) to 512 bytes (i.e. $K = 8$) while T is fixed. It is assumed that the super-segment consists of T = 4 spatially adjacent segments. It is observed that varying the segment size directly influences the row buffer/tag-cache hit rate because it affects the Num_SAB_Row parameter, as shown in Table 6.

As depicted in Fig. 16, increasing the segment size (K) proportionately increases the row buffer and the tag-cache/DAT hit rates. As a result, the average LLC read hit latency is reduced (cf. Fig. 16). On the downside, a larger value of K negatively effects the LLC miss rate (Fig. 16). Increasing K from 1 to 8 increases the miss rate from 18.2% to 23.1%. We can see that configuration with $K = 1$ is optimized for LLC miss rate at the cost of considerably increased LLC read hit latency. Likewise, $K = 8$ is optimized for LLC read hit latency at the expense of worsened LLC miss rate. Our evaluation results show that $K = 4$ strikes a good balance between LLC read hit latency and miss rate and provide better results. Compared to the miss rate optimized $K = 1$, it incurs a mild increase (from 18.2% to 20.5%) in the LLC miss rate but achieves a significant 12% reduction in the LLC read hit latency. Compared to the latency optimized $K = 8$, it incurs 6.5% increase in the LLC read hit latency but achieves a noticeable reduction in LLC miss rate (i.e. from 23.1% to 20.5%).

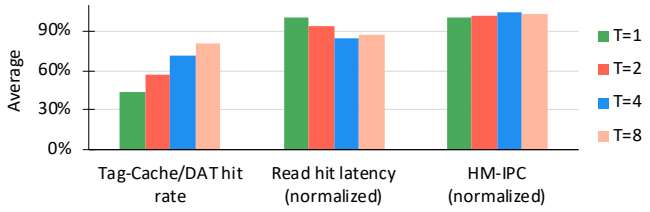

Fig. 17. Impact of the varying super-segment size on various performance metrics ($K = 4$)

4.5.2 Varying the super-segment size

This section presents the impact of varying the super-segment size on various performance metrics and overall performance. For all experiments, a 256 bytes segment size (i.e. $K = 4$) is assumed and the super-segment size is varied from 256 (i.e. $T = 1$) to 2048 bytes (i.e. $T = 8$).

The evaluation results presented in Fig. 17 show that increasing the super-segment size (i.e. T) increases the tag-cache/DAT hit rate. This is due to the fact that more tag columns are fetched after the tag-cache miss (Section 3.1.1). As a result, the number of in-DRAM-tag-comparisons are reduced for a larger T via a high tag-cache hit rate. However, a larger T implies fetching more tag columns after a tag-cache miss which may inflate the LLC read hit latency via increased bus latency. Thus, there is a trade-off between the number of in-DRAM-tag-comparisons (reduces with increasing T) and the number of tag column fetches (increases with increasing T) after a tag-cache miss. While $T = 1$ design reduces the number of tag columns fetches compared to $T = 8$, it exacerbates the tag-cache/DAT hit rate. The $T = 8$ design significantly improves the tag-cache/DAT hit rate but worsens the average LLC read hit latency due to 8 tag columns fetches. Simulation results reveal that $T = 4$ design rescinds the increased tag column fetch effect via higher tag-cache/DAT hit rate and yields better LLC read hit latency.

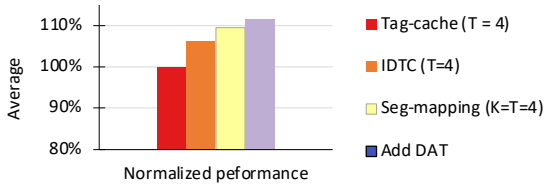

Fig. 18. Impact of individual contribution on performance. IDTC represents the decoupled tag-data configuration with selective in-DRAM-tag comparison for T=4.

4.6 Overall performance

Fig. 18 shows that how each individual policy contributes towards the performance improvement of our DEC-A8 configuration compared to BASE-T$. The decoupled tag-data design achieves 6% performance speedup compared to BASE-T$ via parallel access, selective in-DRAM-tag-comparison, and latency-optimized tag region. Applying intelligent segment mapping (i.e. setting K to 4) adds another 3.5% performance speedup which is achieved via an improved row buffer and tag-cache hit rates without significantly degrading LLC miss rate. Using our DAT provides 2.1% further speedup via reduced in-package bandwidth usage and a higher tag-cache/DAT hit rate.

4.7 Iso-area comparison

Recall from Section 4.4.2 that our 8-way associative DEC-A8 (48 MB tag region and 512 MB data region) outperforms

TABLE 7
Comparisons of configuration parameters for the tag and data regions employing 384 MB DRAM LLC

Parameters	Tag-Region-ISO	Data-Region-ISO
# banks	8	32
# Rows in a bank	4096	4096
# Mats in a bank	128	128
Mat Size	512 x 576	1024 x 768
Row size	1152 Bytes	3072 Bytes
Total size	36 MB	384 MB
Total area	20.9 mm^2	84.7 mm^2
t_{RAS}-t_{RCD}-t_{RP}-t_{CAS}-t_{WR}	$16-4-4-4-4-4(ns)$	$22-5.5-5.5-5.5-5.5(ns)$

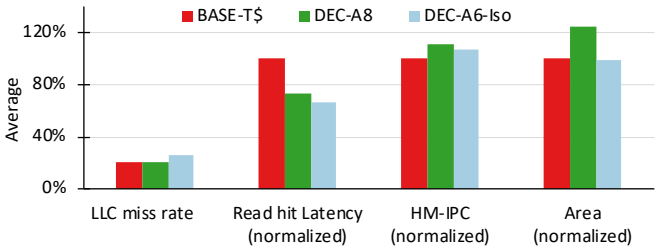

Fig. 19. Performance comparison of the iso-area *decoupled cache* (DEC-A6-ISO) with DEC-A8 and BASE-T$

7-way associative BASE-T$ (512 MB storing tags and data in the same region) by 11.6%. However, this performance improvement comes at the expense of 24.7% area increase (see Fig. 19). This additional area is required for the 48 MB tag region (cf. Table 2). To stay within the same area budget, we evaluate an iso-area decoupled design namely DEC-A6-ISO ($K = 4$ and $T = 4$). Our 6-way associative DEC-A6-ISO employs a 384 MB data region and a 36 MB tag region. Due to lower associativity and higher K, the DRAM LLC miss rate jumps from 21% (in BASE-T$) to 26% (in our DEC-A6-ISO). However, this miss rate increase is compensated by a significant 33.5% reduction in the LLC read hit latency (see Fig. 19). The reason of the LLC read hit latency improvement in our decoupled design compared to BASE-T$ is already discussed in Section 4.4.2. Compared to our larger-area DEC-A8, our DEC-A6-ISO degrades the performance by 4.3% due to increase in the LLC miss rate. Please note that the latency parameters of both the tag and data regions of DEC-A6-ISO (Table 7) are reduced compared to the larger area DEC-A8 (Table 2).

4.8 Iso-area comparison with state-of-the-art

This section provides a detailed comparison of DEC-A6-ISO with the following related works:

1) **SectorCache:** An 8-way sector cache [22] with 512 bytes sector and 64 byte block (Section 2.4 and Fig. 3) equipped with a tag-cache. A single entry of the tag-cache stores the tags of recently accessed 8 sectors.
2) **TIMBER:** A direct-mapped TIMBER design [15] equipped with a tag-cache (Section 2.3 and Fig. 2b). For this design, it is assumed that 4 tag columns (i.e. $T = 4$ and 28 tags) are fetched to update a tag-cache entry following a tag-cache miss.
3) **ACCORD-A4:** The recent ACCORD design [23] with 4-way associativity.

All of the aforementioned configurations implement LLC bypassing. Fig. 20 shows that DEC-A6-ISO outperforms SectorCache, TIMBER, and ACCORD by 11%, 7.5%, and 4.7% respectively. Our DEC-A6-ISO provides the following two latency optimizations which are lacking in these designs. First, Compared to them, our DEC-A6-ISO does not incur t_{WR} latency for the data region in the scenario when the victim row buffer of the data region is not modified. Second, our design provides support for selective in-DRAM-tag-comparison and parallel tag-data accesses. In addition, DEC-A6-ISO outperforms all designs in terms of miss rate as explained in the following.

The main advantage of DEC-A6-ISO compared to TIMBER is that it reduces the LLC miss rate from 33% to 26% due to high associativity (Fig. 20). Each 4 tag column in TIMBER stores the tags of 28 spatial adjacent direct-mapped sets. Therefore, a tag-cache entry in TIMBER provides the hit/miss information of 28 spatial adjacent direct-mapped sets. The TIMBER maps 56 consecutive memory blocks to the same DRAM row (i.e. $Num\_SAB\_Row = 56$; Table 5) which provides significantly high tag-cache hit rate (i.e. 78%) compared to DEC-A6-ISO (70%; $Num\_SAB\_Row = 32$) as shown in Fig. 14. For the same reason, the row buffer hit rate of TIMBER is also high compared to DEC-A6-ISO (Fig. 20). The latency benefits of a high row buffer and tag-cache hit rate is largely compensated by aforementioned latency optimizations. Therefore, the latency of TIMBER is slightly better compared to DEC-A6-ISO as depicted in Fig. 20.

The row organization of ACCORD [23] is similar to BEAR (Fig. 2a) with the exception that ACCORD implements an A-way associativity by combining A TAD entries to constitute a cache set. Note that a tag-cache is not employed for ACCORD-A4 because updating an entry in the tag-cache will require 28 column fetches. ACCORD-A4 steers a cache line to at-most 2-ways to reduce the miss cost of a tag lookup in a 4-way LLC. In addition, ACCORD-A4 applies an intelligent way prediction among 2-ways to reduce the number of tag lookups in case of an LLC hit. ACCORD-A4 outperforms BEAR in terms of miss rate at the cost of slight increase in the hit latency. However, restricting cache lines to 2-ways in ACCORD-A4 causes more conflict misses compared to our DEC-A6-ISO (cf. Fig. 20). In addition, ACCORD-A4 always requires a tag lookup for each LLC-bypassed block in the combined tag-data region. In contrast, for a tag-cache or DAT hit, our DEC-A6-ISO does not require any access to either the tag or the data regions for the LLC-bypassed block. These factors along with the aforementioned latency optimizations provide a latency reduction of 5.9% compared to ACCORD-A4.

The SectorCache employs a large sector size (i.e., 512 bytes) to reduce the overhead of the tag storage and to improve the tag-cache hit rate. A segment size of 512 bytes (i.e., K = 8) and a single tag column per row (i.e., T_{row}) implies that $Num\_SAB\_Row = K$ x $T_{row} = 8$ for SectorCache. In addition to latency optimizations described above, DEC-A6-ISO also experiences a high row buffer hit rate (cf. Fig. 20) and tag-cache hit rate (cf. Fig. 14) compared to SectorCache.

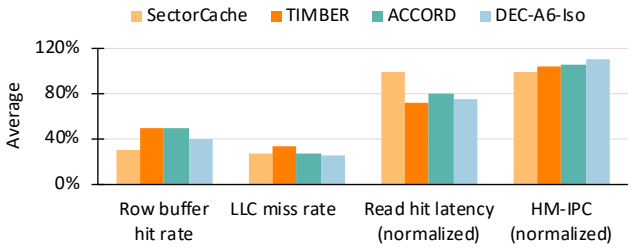

Fig. 20. Comparisons of different designs in terms of various performance metrics

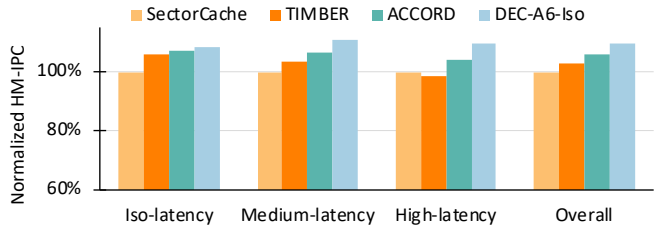

Fig. 21. Normalized average performance for various latencies

This is due to a high value Num_SAB_Row parameter for DEC-A6-ISO (cf. Table 5). As a result, DEC-A6-ISO reduces the latency by 25% compared to SectorCache (cf. Fig. 20). Another disadvantage of the sector cache organization is that it suffers from inefficient resource allocation due to internal fragmentation. This is because it reserves the space for the entire sector, while some of the blocks belonging to a sector may not be referenced before the sector gets evicted from the LLC. Therefore, SectorCache in Fig. 3 incurs a higher DRAM cache miss rate (cf. Fig. 20) compared to DEC-A6-ISO. The simultaneous improvement in LLC read hit latency and miss rate results in performance improvement of 11% compared to SectorCache as shown in Fig. 20.

4.9 Impact of latency assumptions

We further evaluate the aforementioned DRAM designs under the following latency assumptions:

1) **Iso-latency:** The latency of on-chip DRAM cache is similar to that of off-chip memory as assumed in [12], [13].
2) **Medium-latency:** The off-chip memory is assumed to be 1.5x slower compared to on-chip DRAM cache as depicted in Table 3.
3) **High-latency:** The off-chip memory is 4x slower compared to on-chip DRAM cache as assumed in [23].

In all cases, the bandwidth of on-chip DRAM is assumed to be 8x higher than that of off-chip memory. Previous results (cf. Figs 12 to 20) correspond to the medium-latency scenario. Fig. 21 shows the performance of all designs under different latency assumptions. As shown, under the Iso-latency assumption, the performance improvement of DEC-A6-ISO is less prominent compared to TIMBER and ACCORD-A4 because the applications benefit more from lower LLC read hit latency and are not affected severely by LLC miss rate. Therefore, the miss rate improvement of DEC-A6-ISO has less impact on the overall performance. In contrast, under the High-latency assumption, DEC-A6-ISO shows greater speedup because ACCORD and, in particular, TIMBER access high-latency off-chip memory more frequently. In this scenario, the performance of applications severely depends on the LLC miss rate. DEC-A6-ISO outperforms all designs under all latency assumptions since it provides simultaneous reduction in the read hit latency and LLC miss rate.

5 Conclusions

This paper presents a *tag-data decoupled cache* design that not only optimizes LLC read hit latency compared to contemporary associative designs but also ensures the LLC miss rate benefits of associativity compared to existing direct-mapped designs. The proposed decoupled design diminishes the LLC read hit latency using a combination of techniques. First, it enables concurrent access to the tag and data regions, hence reduces the tag serialization latency. Second, it directs bypass accesses to the latency optimized tag region which obviates access to the high latency data region. Third, it provides improved row buffer and tag-cache/DAT hit rates which is achieved by judicious selection of the segment and super-segment sizes. Fourth, the selective in-DRAM-tag-comparison reduces contention in the tag region. Finally, the DRAM bypass policy reduces LLC contention by reducing the number of dead line fills. We evaluate our decoupled design for various SPEC2006 benchmarks and compare the results with state-of-the-art approaches. We show that our iso-area *decoupled cache* design improves the average performance by 11.7% and 7.2% respectively compared to existing associative and direct-mapped DRAM LLC designs.

Acknowledgments

This work was partially funded by the German Research Council (DFG) through the TraceSymm project CA 1602/4-1 and the DART-HMS project HA 9123/1-1.

References

[1] C. Weis, M. Jung, and N. Wehn, *3D Stacked DRAM Memories*. John Wiley & Sons, Ltd, 2019, ch. 8, pp. 149–186. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9783527697052.ch8

[2] J. Kim and Y. Kim, "Hbm: Memory solution for bandwidth-hungry processors," in *2014 IEEE Hot Chips 26 Symposium (HCS)*, Aug 2014.

[3] D. U. L. et al., "A 1.2V 8Gb 8-channel 128GB/s High-bandwidth Memory (HBM) Stacked DRAM with Effective Microbump I/O Test methods using 29nm Process and TSV," in *International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, Feb 2014.

[4] U. e. a. Kang, "8 Gb 3-D DDR3 DRAM using Through-Silicon-Via Technology," in *IEEE Journal of Solid State Circuits*, vol. 45, no. 1, January 2010.

[5] X. Yu, C. J. Hughes, N. Satish, O. Mutlu, and S. Devadas, "Banshee: Bandwidth-efficient DRAM Caching via Software/Hardware Cooperation," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-50 '17. New York, NY, USA: ACM, 2017.

[6] X. Jiang, N. Madan, L. Zhao, M. Upton, R. Iyer, S. Makineni, D. Newell, Y. Solihin, and R. Balasubramonian, "CHOP: Adaptive Filter-Based DRAM Caching for CMP Server Platforms," in *16th IEEE Symposium on High-Performance Computer Architecture (HPCA)*, 2010.

[7] X. Jiang, N. Madan, L. Zhao, M. Upton, R. Iyer, S. Makineni, D. Newell, Y. Solihin, and R. Balasubramonian, "CHOP: Integrating DRAM Caches For CMP Server Platforms," *IEEE Micro Magazine (Top Picks), IEEE Computer Society*, 2011.
[8] D. Jevdjic, S. Volos, and B. Falsafi, "Die-stacked DRAM caches for Servers: Hit Ratio, Latency, or Bandwidth? Have it All with Footprint Cache," in *Proceedings of the 40th International Symposium on Computer Architecture (ISCA)*, 2013.
[9] J. et al., "Unison cache: A scalable and effective die-stacked dram cache," in *47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2014.
[10] Y. Lee, J. Kim, H. Jang, H. Yang, J. Kim, J. Jeong, and J. W. Lee, "A Fully Associative, Tagless DRAM Cache," *SIGARCH Computer Architecture News*, vol. 43, June 2015.
[11] F. Hameed and J. Castrillon, "Rethinking On-chip DRAM Cache for Simultaneous Performance and Energy Optimization," in *19th conference on Design, Automation and Test in Europe (DATE)*, March 2017.
[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.
[13] C. Chou, A. Jaleel, and M. K. Qureshi, "BEAR: Techniques for Mitigating Bandwidth Bloat in Gigascale DRAM Caches," in *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, ser. ISCA '15, 2015.
[14] C.-C. Huang and V. Nagarajan, "ATCache: Reducing DRAM Cache Latency via a Small SRAM Tag Cache," in *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2014.
[15] J. Meza, J. Chang, H. Yoon, O. Mutlu, and P. Ranganathan, "Enabling Efficient and Scalable Hybrid Memories Using Fine-Granularity DRAM Cache Management," *IEEE Computer Architecture Letters*, vol. 11, July 2012.
[16] F. Hameed, L. Bauer, and J. Henkel, "Reducing Latency in an SRAM/DRAM Cache Hierarchy via a Novel Tag-Cache Architecture," in *51st Design Automation Conference (DAC'14)*, 2014.
[17] F. Hameed, L. Bauer, and J. Henkel, "Architecting On-Chip DRAM Cache for Simultaneous Miss Rate and Latency Reduction," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, April 2016.
[18] G. Loh and M. Hill, "Efficiently Enabling Conventional Block Sizes for Very Large Die-stacked DRAM Caches," in *44th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2011.
[19] G. Loh and M. Hill, "Supporting Very Large DRAM Caches with Compound Access Scheduling and MissMaps," *IEEE Micro Magazine, Special Issue on Top Picks in Computer Architecture Conferences*, vol. 32, 2012.
[20] F. Hameed, L. Bauer, and J. Henkel, "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.
[21] H. Yoon, J. Meza, R. Ausavarungnirun, R. Harding, and O. Mutlu, "Scalable high performance main memory system using phase-change memory technology," in *Proceedings of the 30th IEEE International Conference on Computer Design*, May 2012.
[22] Zhao Zhang, Zhichun Zhu, and Xiaodong Zhang, "Design and Optimization of Large size and Low Overhead Off-chip Caches," *IEEE Transactions on Computers*, vol. 53, 2004.
[23] V. Young, C. Chou, A. Jaleel, and M. K. Qureshi, "ACCORD: Enabling Associativity for Gigascale DRAM Caches by Coordinating Way-Install and Way-Prediction," in *45th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2018, Los Angeles, CA, USA, June 1-6, 2018*, 2018.
[24] F. Hameed, L. Bauer, and J. Henkel, "Adaptive Cache Management for a Combined SRAM and DRAM Cache Hierarchy for Multi-Cores," in *Proceedings of the 15th conference on Design, Automation and Test in Europe (DATE)*, March 2013.
[25] K. Yang, H. Tsai, C. Li, P. Jendra, M. Chang, and T. Chen, "etag: Tag-comparison in memory to achieve direct data access based on edram to improve energy efficiency of DRAM cache," *IEEE Trans. on Circuits and Systems*, vol. 64-I, 2017.
[26] F. Hameed and M. B. Tahoori, "Architecting STT Last-Level-Cache for Performance and Energy Improvement," in *17th International Symposium on Quality Electronic Design (ISQED)*, March 2016.
Cache," in *IEEE International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'13)*, 2013.
[27] F. Hameed, L. Bauer, and J. Henkel, "Reducing Inter-Core Cache Contention with an Adaptive Bank Mapping Policy in DRAM
[28] A. Jaleel, W. Hasenplaugh, M. Qureshi, J. Sebot, S. Steely Jr., and J. Emer, "Adaptive Insertion Policies for Managing Shared Caches," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2008.
[29] Y. Xie and G. H. Loh, "Dynamic Classification of Program Memory Behaviors in CMPs," in *2nd Workshop on Chip Multiprocessor Memory Systems and Interconnects (CMP-MSI)*, June 2008.
[30] M. Qureshi, A. Jaleel, Y. Patt, S. Steely, and J. Emer, "Set-Dueling-Controlled Adaptive Insertion for High-Performance Caching," *IEEE Micro Magazine (Top Picks), IEEE Computer Society*, vol. 28, 2008.
[31] E. Vasilakis, V. Papaefstathiou, P. Trancoso, and I. Sourdis, "Decoupled Fused Cache: Fusing a Decoupled LLC with a DRAM Cache," *TACO*, vol. 15, 2019.
[32] O. Naji, C. Weis, M. Jung, N. Wehn, and A. Hansson, "A high-level dram timing, power and area exploration tool," in *2015 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, July 2015.
[33] M. Poremba, T. Zhang, and Y. Xie, "Nvmain 2.0: A user-friendly memory simulator to model (non-)volatile memory systems," *IEEE Computer Architecture Letters*, vol. 14, July 2015.
[34] A. A. Khan, F. Hameed, and J. Castrillon, "Nvmain extension for multi-level cache systems," in *Proceedings of the 10th Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools, co-located with 13th International Conference on High-Performance and Embedded Architectures and Compilers (HiPEAC)*, January 2018.
[35] S. Rixner, W. Dally, U. Kapasi, P. Mattson, and J. Owens, "Memory Access Scheduling," in *Proceedings of the 32nd International Symposium on Computer Architecture (ISCA)*, June 2000.
[36] "SAFARI Group at CMU," http://www.ece.cmu.edu/~safari/, 2018, [Online; accessed 25-March-2018].
[37] J. L. Henning, "SPEC CPU2006 Benchmark Descriptions," *SIGARCH Computer Architecture News*, vol. 34, September 2006.

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 the Institute of Space Technology in 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.

Asif Ali Khan Asif Ali Khan is a doctoral researcher at the Chair for Compiler Construction in the Computer Science Department of the TU Dresden, Germany. His current research interests include: Computer architecture, heterogeneous memories, and compiler support for memory system.

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.