Performance and Energy-Efficient Design of STT-RAM Last-Level Cache

Fazal Hameed, Asif Ali Khan, and Jeronimo Castrillon

***Abstract*— Recent research has proposed having a die-stacked last-level cache (LLC) to overcome the memory wall. Lately, spin-transfer-torque random access memory (STT-RAM) caches have received attention, since they provide improved energy efficiency compared with DRAM caches. However, recently proposed STT-RAM cache architectures unnecessarily dissipate energy by fetching unneeded cache lines (CLs) into the row buffer (RB). In this paper, we propose a selective read policy for the STT-RAM which fetches those CLs into the RB that are likely to be reused. In addition, we propose a tags-update policy that reduces the number of STT-RAM writebacks. This reduces the number of reads/writes and thereby decreases the energy consumption. To reduce the latency penalty of our selective read policy, we propose the following performance optimizations: 1) an RB tags-bypass policy that reduces STT-RAM access latency; 2) an LLC data cache that stores the CLs that are likely to be used in the near future; 3) an address organization scheme that simultaneously reduces LLC access latency and miss rate; and 4) a tags-to-column mapping policy that improves access parallelism. For evaluation, we implement our proposed architecture in the Zesto simulator and run different combinations of SPEC2006 benchmarks on an eight-core system. We compare our approach with a recently proposed STT-RAM LLC with subarray parallelism support and show that our synergistic policies reduce the average LLC dynamic energy consumption by 75% and improve the system performance by 6.5%. Compared with the state-of-the-art DRAM LLC with subarray parallelism, our architecture reduces the LLC dynamic energy consumption by 82% and improves system performance by 6.8%.**

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

I. Introduction

CONVENTIONAL off-chip memories do not fulfil the bandwidth and latency requirements of complex applications running on multicore systems [1]. The limited number of I/O pins provided by the packaging induces a gap between processor and memory performance. This widening gap is known as the memory wall [2] and severely limits the performance of applications with large memory and bandwidth requirements. A potential solution to mitigate the memory wall problem is to employ die-stacked technologies [3]–[5]. These technologies integrate processor and memory dies by employing a low-latency and high-bandwidth interconnect. To reduce the number of high-latency accesses to bandwidth-limited off-chip memory, recent research proposed to implement the die-stacked memory as a large-capacity last-level cache (LLC) [6]–[13].

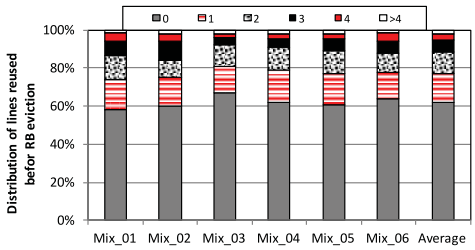

Fig. 1. Distribution of number of unique lines reused before RB eviction using 2-KB RB size for SPEC2006 application mixes (see Table II).

Previously, on-chip DRAM memory has been adopted as an LLC for performance improvement due to its capacity advantage compared with an area equivalent SRAM cache [6]–[10]. However, the DRAM LLC dissipates a significant portion of chip power budget due to its high refresh rate and associated dynamic energy consumption. Therefore, recent research has advocated the use of nonvolatile spin-transfer-torque random access memory (STT-RAM) as LLC [11], [14]. By exploiting the nonvolatility characteristics of STT-RAM, the energy consumption of the LLC can be reduced significantly compared with DRAM.

Typically, a high-capacity STT-RAM holds multiple banks, and each bank is provided with a row buffer (RB) [11], [15]. The total energy consumption in the existing STT-RAM LLC is exacerbated by reading data from an STT-RAM bank into the RB [11]. To reduce the energy consumption of the STT-RAM LLC, we exploit the fact that most of the cache lines (CLs) are unnecessarily fetched into the RB as they are not likely to be reused in the near future. Fig. 1 shows that the probability for the RB content to be accessed before RB eviction is less than 40%. Therefore, we fetch only those CLs into the RB that are likely to be reused. This significantly reduces the number of RB fetches and, thereby, reduces the dynamic energy consumption of an STT-RAM LLC. However, identifying the likelihood for a CL to be reused induces a high tag access latency when using the existing tag read policy [9]–[11] and tag organization. Therefore, we propose

Manuscript received August 14, 2017; revised December 8, 2017; accepted January 20, 2018. This work was supported in part by the German Research Council through the Cluster of Excellence Center for Advancing Electronics Dresden. *(Corresponding author: Fazal Hameed.)*

F. Hameed is with the Chair for Compiler Consruction, Technische Universität Dresden, 01069 Dresden, Germany and also with the Institute of Space Technology, Islamabad 44000, Pakistan (e-mail: fazal.hameed@tu-dresden.de).

A. A. Khan and J. Castrillon are with the Chair for Compiler Consruction, Technische Universität Dresden, 01069 Dresden, Germany (e-mail: asif_ali.khan@tu-dresden.de; jeronimo.castrillon@tu-dresden.de).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2018.2804938

a novel RB tags-bypass policy and a novel tag organization that provides fast access to LLC tags. To reduce LLC latency via access parallelism, we modify the mapping of tags to the columns of an STT-RAM array. More precisely, we make the following contributions.

1) We classify lines of the STT-RAM row into highly reused and lowly reused lines.
2) We propose a selective read policy that only fetches highly reused lines of an STT-RAM row into the RB. This significantly reduces the energy consumption.
3) While the selective read policy saves energy, it also induces a latency penalty. To counteract this effect, we propose an RB tags-bypass policy.
4) We propose a small LLC data cache (LDC) that stores the lines which are likely to be accessed in near future. Lines that hit in the LDC are accessed with a much lower latency.
5) We propose a tag organization that reduces the number of STT-RAM column accesses and the LLC miss rate which provides simultaneous performance and energy benefits.
6) A novel tags-update policy is proposed which saves energy by reducing the number of tag writebacks.
7) We propose a tag-to-column mapping policy that improves access parallelism by allowing simultaneous accesses to different subarrays. Unlike contemporary approaches, we exploit subarray parallelism to its full potential by uniformly distributing requests to different subarrays.

The rest of this paper is organized as follows. Section II presents an overview of the LLC organizations and the working principle of STT-RAM and DRAM. Section III describes the proposed architecture. Experimental setup, evaluation results, and comparison with state-of-the-art proposals are detailed in Section IV. Section V presents the related work followed by conclusion in Section VI.

II. Background

This section describes the organization of the recently proposed LLCs, the basic operating principles of the STT-RAM, and its qualitative comparison with the DRAM.

A. State-of-the-Art LLC Organization

Fig. 2(a) illustrates the organization of the previously proposed LLC [7], [10], [11]. The LLC is split into multiple memory banks each equipped with an RB. Each bank is organized as a series of rows that are split into columns. In this paper, we assume an LLC size of 256 MB, a row size of 2048 bytes (2 KB), and a column size of 64 bytes.

The LLC is implemented as a set-associative cache. Recent research has proposed various set mapping policies [7], [9]–[11], [16]. This section explains LAMOST, which is the most recently proposed policy, and has been used for the DRAM [7], [10] LLCs and the STT-RAM LLC [11]. In LAMOST, the tags and the lines of an LLC are stored in the same row. Each 2-KB LLC row comprises four sets with seven-way associativity. A set contains seven CLs and a tag column. Each tag column and the CL have a size of 64 bytes.

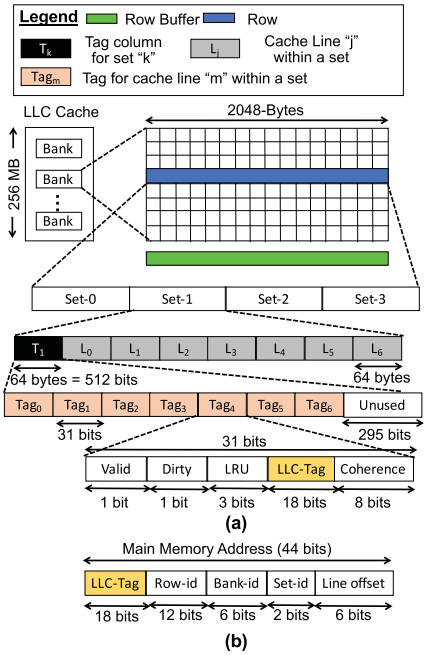

Fig. 2. Typical LLC and address organization using the LAMOST policy [7], [10], [11] for an eight-core system with 256-MB LLC size, 64 banks, 2-KB RB size, and 44-bit physical addresses. (a) LLC LAMOST organization. (b) Address organization.

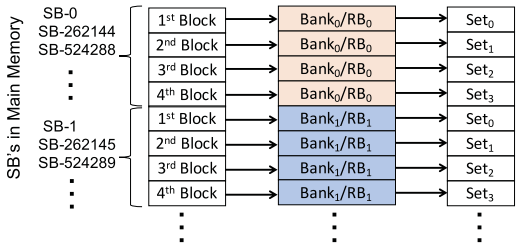

Fig. 3. Bank/row/set mapping for LAMOST policy [7], [10], [11]. SB stands for super block.

A tag column stores seven 31-bit wide tag entries, thus using only $7 \times 31 = 217$ bits. This leaves 295 bits of each tag column unused.

We assume 44-bit physical address which is split into multiple parts as depicted in Fig. 2(b). The 18 most significant bits identify the LLC tag. The next 12 bits select a row within a bank. The following 6 bits select a memory bank within the LLC. The next 2 bits are used to identify the set within a row, and the remaining six least significant bits identify a byte within a CL. Fig. 3 shows the mapping of main memory blocks to a particular bank, row, and set using the LAMOST policy. All the memory blocks that are mapped to the same row are referred to as a superblock (SB). As evident from Fig. 3, each SB consists of four consecutive 64-byte memory blocks.

For each LLC bank, the RB holds a copy of the row that was retrieved last. In order to access a CL, the controller issues a command to fetch the corresponding row in the bank to its RB. Successive requests to the CLs of the same row are serviced faster without fetching the row again. This is referred to as an RB hit. An access to a line in a different row requires to fetch the corresponding row into the RB. This is referred to as an RB miss. The access latency and energy of an RB hit is much lower compared with that of an RB miss.

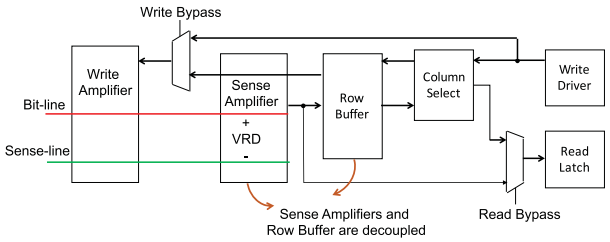

Fig. 4. STT-RAM peripheral circuit that supports both memory access via an RB and direct access to the STT-RAM array via a bypass [11], [15].

Therefore, an application with high RB locality (many RB hits) has a better performance and a lower energy consumption than a similar application with low RB locality.

B. Basic Operating Principles of DRAM and STT-RAM

In DRAM, data are stored as a charge in a bit cell. During a DRAM cell read operation, the charge stored in the capacitor is shared with the bitline. The sense amplifier that is connected to each bitline detects the voltage change which is then translated to either logical "0" or logical "1." In the DRAM, all the sense amplifiers that are connected to the bitlines are referred to as the RB. Due to this physical connection, the RB and the bit cells in the DRAM bank are tightly coupled.

An STT-RAM bit-cell uses a magnetic tunnel junction (MTJ) device to store the information. The MTJ is made of two independent ferromagnetic layers, the reference layer, and the free layer. The magnetic orientation of the reference layer is fixed. However, the magnetic orientation of the free layer can be freely rotated. It can be parallel or antiparallel to the reference layer. Depending on the magnetic orientation of the free layer, the resistance of the MTJ cell changes. In the parallel state, the resistance is low and the cell stores a logical "0." In the antiparallel state, the resistance is high and the cell stores a logical "1."

C. STT-RAM RB Bypass and Partial Write Optimizations

In the DRAM, the RB (i.e., sense amplifiers) and the bit-cells are tightly coupled due to charge sharing. Therefore, any read/write operation in the DRAM requires the data to be fetched into the RB. A prominent characteristic of the STT-RAM is the decoupled organization [11], [15], [17] of its sense amplifiers and the RB (Fig. 4). This is a major advantage over the DRAM for the following three reasons.

1) It is possible to bypass the RB in the STT-RAM. The read or write operation can be performed directly on the STT-RAM bank bit cells without the need to fetch the data into the RB. This RB bypassing has been leveraged recently [11], [15] to improve energy efficiency.
2) Read or write operations can be performed directly on the RB without requiring the sense amplifiers. An STT-RAM bank column access does not necessarily require the RB.
3) The STT-RAM bank and the RB can operate independently on different rows which improves parallelism.

Traditionally, on an RB miss, the newly requested row is always fetched in the RB. However, if the row that is currently stored in the RB is likely to experience RB hits, it should not be replaced by a row that is likely to experience an RB miss. Therefore, the traditional policy does not work well due to the following reasons. First, eviction of useful rows (i.e., that are likely to experience RB hits) reduces the RB hit rate and therefore reduces the overall performance. Second, a low RB hit rate leads to a large number of RB fetches which significantly increases the energy consumption.

Considering the cache access types (read, write, and write-back), the writeback access is particularly worse in terms of performance. An access after a writeback has a chance of less than 5% to hit in the RB [11]. Therefore, recent work [11], [15] has proposed to bypass the RB for the writeback access and to perform the cache writeback operation directly on an STT-RAM column. This improves both performance and energy efficiency by increasing the RB hit rate.

To further reduce the energy consumption, the partial write approach proposed in [15] writes only dirty columns from the RB back to the STT-RAM array after an RB eviction.

D. Small Row/RB Sizes

An STT-RAM bank is typically realized as a combination of multiple subarrays each of which can operate independently [17] by making trivial changes in the access scheduler [18]. Conceptually, the large STT-RAM row and the RB can be divided into multiple small subrows and sub-RBs, respectively, as shown in Fig. 13. This so-called small RB organization allows to perform only subrow (e.g., activate) and sub-RB level operations (e.g., read and write). The advantages of employing small RB organization (e.g., 512-bytes row/RB size) compared with the large RB organization (e.g., 2048-bytes row/RB size) are reduced energy consumption and improved subarray parallelism [18]. However, employing the small RB organization using the existing LLC designs [7], [9]–[12] leads to performance inefficiencies due to reduced subarray parallelism. To exploit the full potential of subarray parallelism using small RB organization, we redesigned LLC organization that uniformly distributes requests to different subarrays of the same bank.

III. Proposed STT-RAM LLC Architecture

Based on the state-of-the-art discussed in Section II, this section discusses our novel STT-RAM-based LLC architecture.

A. Overview

Fig. 5 depicts an overview of the STT-RAM-based LLC architecture and highlights our contributions. Similar to [11], [15], the proposed architecture uses RB bypass and partial write optimizations (recall Section II-C) to improve performance and energy consumption. A disadvantage of the existing LLC RB policy [7], [9]–[11], [16] is that it always fetches an entire row into the RB. We introduce selective read policy that reads only highly reused STT-RAM columns into the RB and thereby reduces energy consumption. However, with the selective read policy, the existing tag read policy

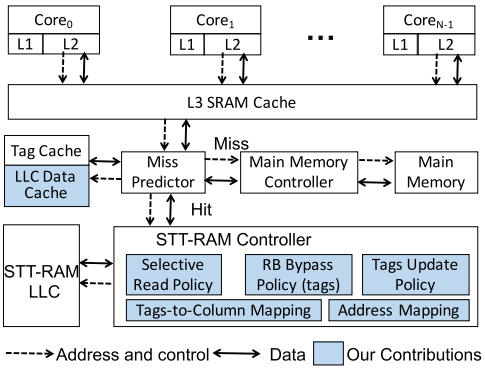

Fig. 5. STT-RAM-based LLC architecture for an N-core system highlighting our contributions.

incurs high access latencies. To address this problem, we propose a novel RB tags-bypass policy and an LDC that cut down the access latency and improve the LLC performance. Furthermore, we present a novel address organization and a tags-to-column mapping policy that improves the RB hit rate and subarray parallelism. To reduce energy consumption, we propose efficient tag organization and tags update policy that simultaneously reduce the number of tag columns reads and writes.

B. Selective Read Policy

The basic tenet of our proposed selective read policy is to reduce STT-RAM energy costs by eliminating unnecessary STT-RAM array column reads. When we fetch only those CLs that are likely to be reused, we avoid unnecessarily fetching of data from the STT-RAM array and thereby reduce the energy consumption.

In order to quantify how often LLC lines in the RB are reused, we conducted a simple experiment. For this, we used a state-of-the-art STT-RAM LLC with 2-KB RB size, the LAMOST set mapping policy, and the STT-RAM optimizations discussed in Section II-C. We executed a series of SPEC2006 [19], [20] application mixes as listed in Table II and counted the number of lines that are reused in the RB before eviction. Fig. 1 displays the results of this experiment and shows that the majority of RB fetches are unnecessary, as none of the lines in the RB is accessed again in over 60% of the cases. The percentage of times that more than four lines in the RB are reused before eviction is less than 2%.

The RB has a low utilization due to the bank/row/set mapping [Fig. 2(b)] of the LAMOST policy as shown in Fig. 3. Since the SBs that are mapped to the sets of the same LLC row are spacial distant, it is unlikely that subsequent accesses hit the RB.

For selective read policy, we classify the LLC lines as highly reused and lowly reused lines. We mark one line for each of the four sets within an LLC row as highly reused. We distinguish two cases for selection. If a memory block of the currently requested SB is present in the set, then the corresponding line is highly reused. This considers the fact that subsequent accesses are likely to be within the same SB. If a set does not contain a memory block of the currently requested SB, the least recently used (LRU) dirty line of this set is marked as highly reused line. This is based on the fact that this line is likely to be replaced on a subsequent access. Then, the line will need to be written back to the main memory if it is dirty. Therefore, it is likely to be reused.

We illustrate the selective read policy using a simple example as illustrated in Fig. 6. We assume an SB S_b that contains four adjacent memory blocks named b_0, b_1, b_2, and b_3. The memory blocks are mapped to the sets Set_0, Set_1, Set_2, and Set_3, respectively. Furthermore, we assume that memory block b_2 is currently requested. On an RB miss, the corresponding row needs to be fetched into the RB. Traditionally, the complete row would be fetched. However, using the selective read policy, we fetch only highly reused lines.

In the given example, the highly reused lines are marked by arrows. The lines L_0 of Set_0 and L_4 of Set_2 are highly reused lines, since they hold the memory blocks b_0 and b_2 of the currently accessed SB S_b.

The other memory blocks of S_b (b_1 and b_3) are not currently present in the sets Set_1 and Set_3. Therefore, the dirty LRU line in Set_1 is marked as highly reused, whereas the clean LRU line in Set_3 is not marked as highly reused. In order to decide which line is highly reused, the four tag columns need to be fetched first. Then, the highly reused lines are fetched into the RB. In the proposed architecture, a total of seven columns, four tag columns, and three highly reused lines (for this particular example) are read into the RB unlike 32 column reads in existing LLC architectures.

If a subsequent request misses the RB, the currently resident row is evicted from the RB and the whole process is repeated for the new request. However, if a subsequent request hits the RB, a check is needed if the requested line is present in the RB. If the line is present in the RB, the request is directly served. Otherwise, the rare case in Fig. 1 is encountered where more than four lines of a row are reused. Subsequently, the entire row from the STT-RAM array is fetched into the RB.

C. Latency Breakdown

In this section, we analyze the latencies that occur in the STT-RAM-based LLC architecture. We assume the LLC architecture depicted in Fig. 5 along with a tag cache. We further assume a request to memory block b_2 as in Fig. 6. The latency breakdown for various scenarios in state-of-the-art LLC architectures is explained as follows.

1) Tag-Cache Miss and RB Miss in LAMOST: Fig. 7(a) shows the latency breakdown of a tag-cache and RB miss without employing the selective read policy. This latency includes 18 cycles for row activation (ACT), 18 cycles for column access latency (CAS) to access the tag column (i.e., T_2), two cycles to transfer the relevant tag column on the bus (i.e., RD T_2), one cycle for the tag check, another 18 cycles to access the requested CL, and two cycles to read the CL (i.e., RD CL). This is a total of 59 cycles. After that, the controller issues successive read request to prefetch the remaining tag columns (i.e., RD T_0, T_1, and T_3) into the tag cache. The additional latency required to read the nonrequested tag columns (i.e., T_0, T_1, and T_3) is six cycles. This extra latency overhead is repayed by future hits in the tag cache,

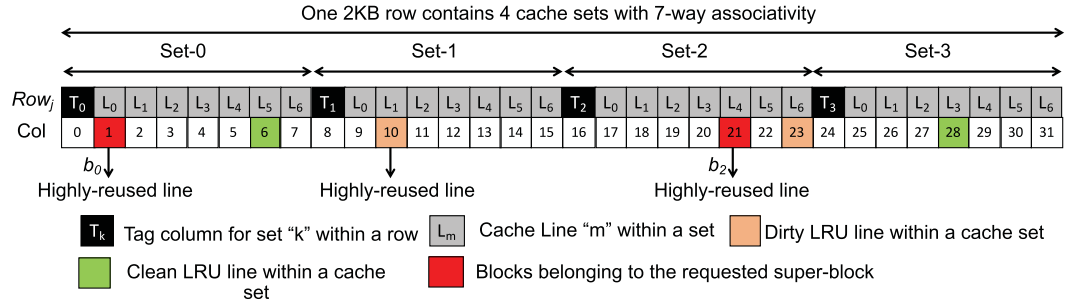

Fig. 6. Example illustrating the selective read policy using LAMOST organization and 2-KB row size.

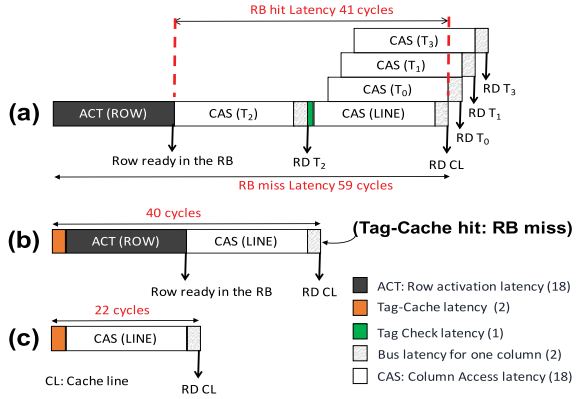

Fig. 7. Latency incurred in LAMOST policy for various scenarios. (a) Tag-cache miss and RB miss. (b) Tag-cache hit and RB miss. (c) Tag-cache hit and RB hit.

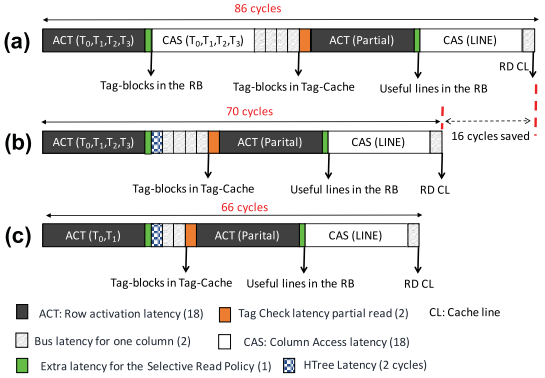

Fig. 8. Tag-cache miss and RB miss latencies with selective read policy using (a) existing tag read policy, (b) controller optimizations exploiting RB bypassing for the tags, and (c) proposed tag organization.

leveraging the temporal locality of the application. Therefore, the RB/tag-cache miss latency is 59 cycles.

2) Tag-Cache Hit and RB Miss: A tag-cache hit does not require an extra CAS command to access the tag column T_2, which is vital to identify the location of the requested line. Therefore, read requests that hit in the tag cache [Fig. 7(b)] has a much smaller latency (i.e., 40 cycles) compared with the tag-cache miss requests.

3) Tag-Cache Hit and RB Hit: Read requests that hit in both the tag cache and the RB do not require an extra row activation and are serviced much faster (i.e., 22 cycles) as illustrated in Fig. 7(c).

D. RB Tags-Bypass Policy

The LLC tag read policy in [6]–[8] and [11] always fetches the tags (and the lines) in the RB before inserting them in the tag cache. However, this policy worsens the RB miss and tag-cache miss latency for the selective read policy as shown in Fig. 8(a). The selective read policy requires two row activations. One for fetching the four tag columns into the RB, and one subsequent activation for fetching the highly reused lines. Between these row activations, additional cycles CAS and bus latencies will be required to read the tag columns from the RB into the tag cache.

In addition, the implementation of the selective read policy requires modification to row activation by adding an extra enable bit to each sense amplifier. This extra bit is required to disable the sense amplifiers for a certain set of columns. Therefore, the selective read policy requires an extra one cycle latency. As a result, the RB miss and tag-cache miss scenario latency is 86 cycles for the selective read policy while using contemporary tag read policy.

The additional latency overhead for an RB and tag-cache miss using the selective read policy is 27 cycles $(86 - 59 = 27)$ when compared with the LAMOST policy. In order to reduce this latency overhead, we propose to bypass the RB for accessing the tags by exploiting the decoupled structure of the sense amplifiers and the RB in the STT-RAM. Instead of fetching the tags into the RB, we store them directly in the tag cache. Fig. 8(b) shows the sequence of commands to read the tags and the CL for this scenario.

The CAS latency (i.e., 18 cycles) comprises three major components. The first latency component is required to load the data from the sense amplifier into the RB. The second latency component is required to read the data from the RB into the read latch. The third latency component is required to move the data out of the bank through the H-tree (i.e., two cycles). The RB tags-bypass policy avoids the first two latency components (i.e., 16 cycles) via bypassing the RB and the column select (see Fig. 4). However, it cannot avoid the third latency component. By avoiding the two latency components of CAS, the latency is reduced by 16 cycles to a total of 70 cycles. Apart from reducing the access latency, the RB tags-bypass policy also reduces energy consumption by avoiding duplicate storage of tags.

To further improve the performance, we update the tags in the tag cache without immediately writing them back to the STT-RAM. Since an STT-RAM write is more expensive than a write to the tag cache, this approach potentially saves energy and reduces latency. On the contrary,

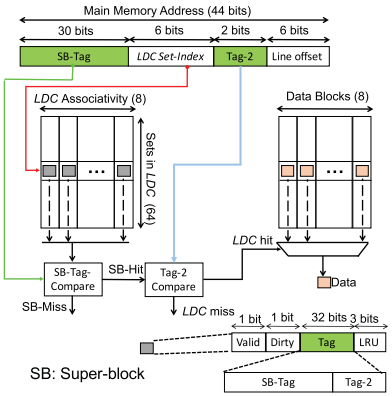

Fig. 9. LDC organization.

TABLE I

LATENCIES FOR READ REQUESTS IN VARIOUS SCENARIOS EXCLUDING CONTROLLER LATENCY

RB Hit	Tag-Cache Hit	Cache Line Latency	Fig.
No	No	70 cycles	8b
No	Yes	40 cycles	7b
Yes	No	41 cycles	7a
Yes	Yes	22 cycles	7c

contemporary LLC architectures originally used for DRAM architectures [7], [9], [10], [12] always update the tags in both the tag cache and the bank array. In our implementation, the tags are only written back to the STT-RAM array when they are evicted from the tag cache.

E. LLC Data-Cache Organization

The access latency of reading a line from the STT-RAM array varies from 22 to 70 cycles depending on the scenario (see Table I). To further reduce the latency, we extend our LLC architecture by a small on-chip SRAM structure called LDC that holds the LLC CLs that are likely to be accessed in the future. Fig. 5 shows the addition of the LDC to the existing LLC architecture.

Fig. 9 illustrates the organization of the LDC. It holds 64 sets with eight-way associativity and has a total size of 34.5 KB. The LLC access that hit the LDC are serviced in only two cycles, due to the small structure. This accelerates LDC hits significantly compared with the 22 to 70 cycles for STT-RAM accesses.

On an LDC access, the LDC set-index field of a main memory address selects a set in the LDC. All tag fields in this set are then compared with the tag fields of the memory address to identify an LDC hit. The SB-Tag field of the memory address corresponds to the SB and the Tag-2 field identifies the memory block within an SB.

Fig. 10 shows the control flow of the LDC controller in our proposed LLC architecture. On each LLC request, the LDC is accessed first in order to identify an LDC hit. If the access hits the LDC, the request is serviced directly, and an STT-RAM access is avoided. Otherwise, the request needs to be serviced regularly by the LLC. New lines are only inserted into the LDC on LLC read hits. In all other cases, the LDC is left unchanged. On an LLC read hit, the currently requested memory block is

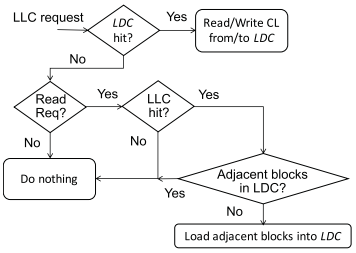

Fig. 10. LDC lookup and insertion policy.

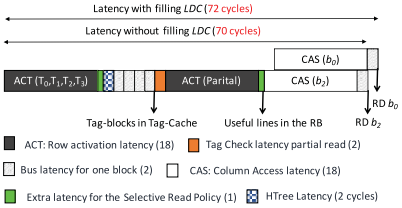

Fig. 11. RB/tag-cache miss latency with LDC using the proposed LDC insertion policy in Fig. 10 for the illustrating example in Fig. 6.

not written to the LDC, but all adjacent memory blocks of the corresponding SB are loaded to the LDC. The proposed LDC insertion policy exploits the temporal and spatial locality of applications as adjacent memory blocks of an SB are likely be accessed in the near future. Note that all the memory blocks of the SB are mapped to the same LDC set.

To elaborate our LDC insertion policy, we use the example from Fig. 6. We assume that b_2 is currently requested and that b_0 does not reside in the LDC. Since the request to b_2 hits the LLC, the memory block is transported to the lower level caches (i.e., L3, L2, and L1) and finally to the requesting core. Since b_2 will be stored in the L1 cache, further requests to b_2 are unlikely to arrive at the LLC. Therefore, b_2 is not inserted into the LDC. However, adjacent memory blocks of the same SB are very likely to be accessed in the near feature. Therefore, all adjacent memory blocks that are present in the LLC are loaded to the LDC. In the example, b_0 is loaded to the LDC after the read request to b_2.

Fig. 11 shows the latency breakdown of an RB and tag-cache miss for the above-mentioned example using the proposed LDC insertion policy in Fig. 10. First, the LLC controller loads the tag columns from an STT-RAM array into the tag cache and identifies the highly reused lines (see Fig. 6) of the row. Then, the highly reused lines are loaded into the RB. Subsequently, read requests are sent to read the memory blocks b_2 and b_0 from the RB. b_2 is forwarded to the L3 cache and b_0 is stored in the LDC. Requesting b_0 in addition to b_2 only incurs an overhead of two cycles as illustrated in Fig. 11. This latency overhead depends upon the number of adjacent memory blocks that are currently present in the LLC. It ranges from 0 to 6 cycles for zero to three present memory blocks, respectively. The latency overhead for reading additional memory blocks to fill LDC is compensated by future hits in the LDC, exploiting the fact that these memory blocks will most likely be accessed later (confer to Fig. 19 for evaluation).

F. Tag Organization

The LLC row in our tag organization is partitioned into two tag columns and 30 CLs as depicted in Fig. 12(b). We employ

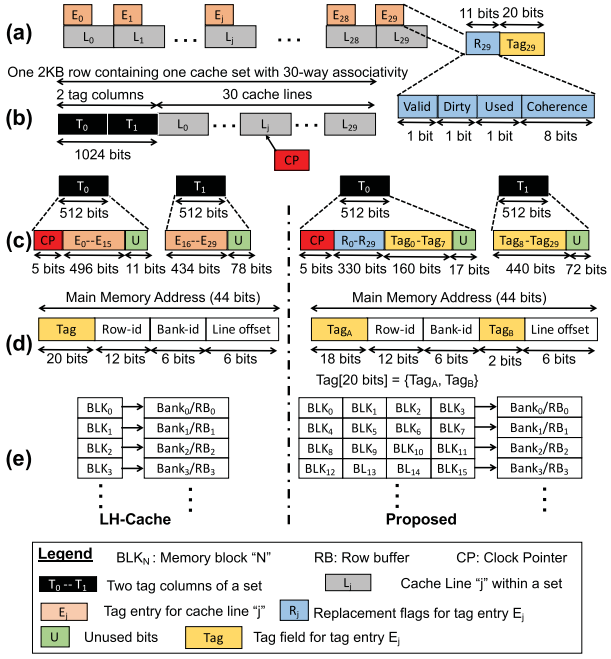

Fig. 12. (a) Logical organization of a cache set with 30-way associative cache. (b) Physical organization of a cache set. (c) Tag organization. (d) Address organization. (e) Memory block to RB mapping. LH-Cache [9], [12] is shown in left column and our proposal shown is shown in right column.

the "pseduo-LRU" [21] cache replacement policy which has reduced storage overhead compared with the traditional LRU policy. The overhead of the pseudo-LRU policy is 31 bits per CL (1 valid bit, 1 dirty bit, 1 used bit, 8 coherence bits for an 8-core system, and 20 tag bits) as shown in Fig. 12(a). The valid bit indicates whether a CL contains a valid or invalid memory block. The dirty bit of CL indicates whether the main memory block has been modified by the processor or remained unchanged, since it was fetched from main memory. To choose a victim CL in a cache set, the pseudo-LRU policy uses a small counter namely "clock pointer" [referred to as CP in Fig. 12(b)]. The CP tracks the current clock position. Inserting a new CL requires to update its tag field and to clear its used bit. After that the CP points to the next CL. The dirty and the used bit are set on CL writeback and CL hit, respectively. On CL eviction, the CL pointed to by the CP is examined. If the used bit of the CL is cleared, it is evicted. Otherwise, the pseudo-LRU policy clears the used bit and gives a second chance to the CL by advancing the CP. It repeats the same check until it finds a CL with a cleared used bit.

In our tag organization, a read request needs an access to two tag columns (in contrast to four tag column accesses in LAMOST) before accessing the useful CLs in selective read policy. Thus, our tag organization saves four cycles for the RB/tag-cache miss latency [Fig. 8(c)] compared with LAMOST [Fig. 8(b)]. In addition, it saves STT-RAM energy consumption compared with LAMOST via reading and updating (upon tag-cache eviction) less number of tag columns. It also reduces LLC miss rate via providing 30-way associativity compared with seven-way associative LAMOST. Our proposal is similar to the state-of-the-art proposed by Loh and Hill [9] namely LH-Cache [12] in a sense that each

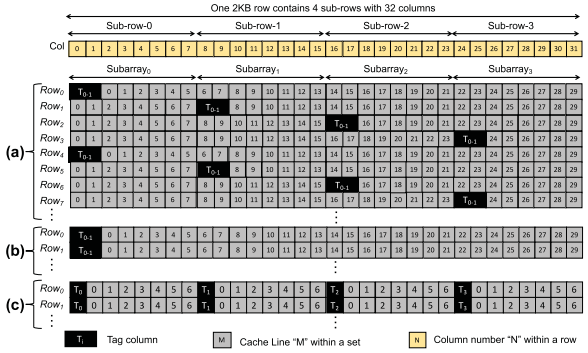

Fig. 13. Tags-to-column mapping. (a) Proposed. (b) LH-Cache [9], [12]. (c) LAMOST [10].

LLC row consists of two tag columns and 30 CLs. However, for performance benefits, it differs from LH-Cache in a number of ways as explained in Sections III-G–III-I and shown in Figs. 12(c)–(e), 13(a) and (b), and 14.

G. Address Organization

The left column of Fig. 12(d) illustrates the address organization of LH-Cache. In their approach, spatially closed memory blocks are mapped to different RBs. For instance, memory blocks BLK_0, BLK_1, BLK_2, and BLK_3 are mapped to RB_0, RB_1, RB_2, and RB_3, respectively, as shown in the left column of Fig. 12(e). In LH-Cache, the likelihood of temporally close requests accessing the same RB, is very low which is why it has a low RB hit rate. To improve the RB hit rate, we modify the address organization which is shown in the right column of Fig. 12(d). In our approach, four consecutive memory blocks are mapped to the same RB. For instance, memory blocks BLK_0, BLK_1, BLK_2, and BLK_3 are mapped to RB_0 as depicted in the right column of Fig. 12(e). The proposed mapping leverages the spatial locality of applications in which the adjacent memory blocks of the current request will most probably be referenced in the near future. As a result, our approach improves the performance and energy consumption via a high RB and tag-cache hit rates as evaluated in Section IV.

H. Tags-Update Policy

LH-Cache and our proposed tag organization differ in the way tag entries are physically arranged in the tag columns as illustrated in Fig. 12(c). Each tag entry E_j of a particular CL L_j in the "pseduo-LRU" policy requires 20-bits tag field and 11 bits for the replacement flags (valid bit, dirty bit, used bits, and coherence bits) to identify hit/miss. These replacement flags (R_j) are modified after a cache hit (set used bit), cache writeback (set dirty bit), and updating coherence bits while the tag field (i.e., Tag_j) remains unchanged. Both the tag field and the replacement flags are modified when a new CL is inserted into the LLC. The replacement flags in our tag organization are stored in the tag column T_0 as illustrated in the right column of Fig. 12(c). In addition, T_0 also stores the tag fields of eight CLs (i.e., Tag_0–Tag_7). The second tag column T_1 stores the tag fields of the remaining 22 cache lines (i.e., Tag_8–Tag_{29}). In our approach, only T_0 needs to

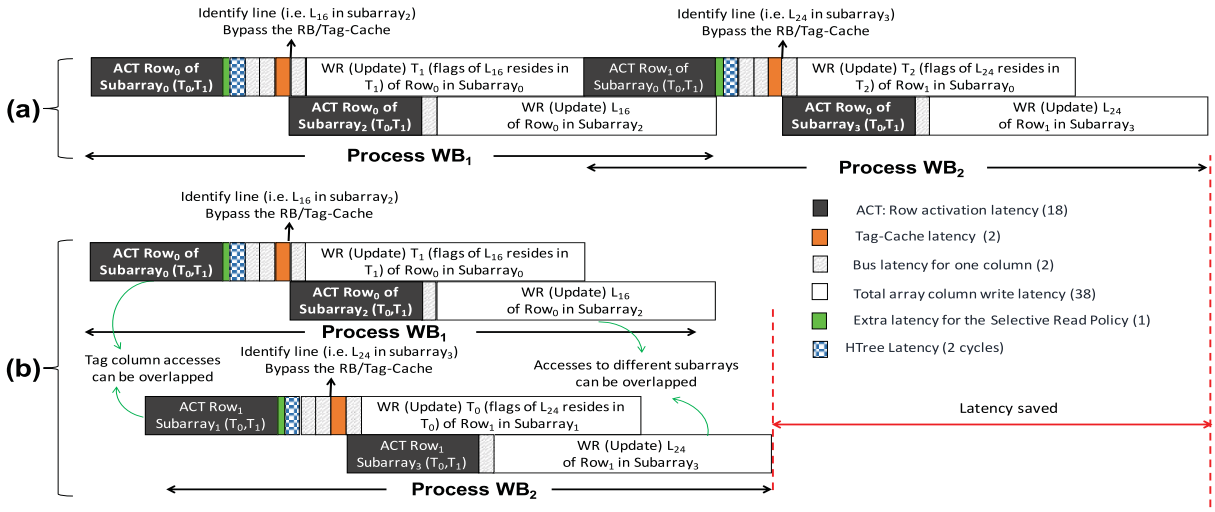

Fig. 14. Service timeline of two writeback requests in (a) LH-Cache and (b) our proposal.

be modified on a cache hit, cache writeback, and on updating coherence information. T_2 needs to be updated only after a new CL insertion into any of the locations L_8–L_{29}. On the contrary, LH-Cache stores the replacement flags and the tag fields of L_0–L_{15} in the tag column T_0 and that of L_{16}–L_{29} in the tag column T_1 [see the left column of Fig. 12(c)].

We illustrate our tags-update policy in comparison with LH-Cache using a simple example. We assume LLC hit requests to three CLs L_3, L_{17}, and L_{20} of a particular row Row_i/Set_i. The first LLC hit request to L_3 will bring the tag columns (T_0 and T_1) of Row_i into the tag cache. Subsequent LLC hit requests to L_{17} and L_{20} will access the tags from the tag cache as it is a tag-cache hit for Row_i. When the tag columns of Row_i are evicted from the tag cache, the replacement flags (i.e., R_3, R_{17}, and R_{20}) need to be updated in the LLC. However, the corresponding tag fields (i.e., Tag_3, Tag_{17}, and Tag_{20}) remain unchanged, because all lines result in the LLC hit. Considering the same example for LH-Cache, both T_0 (i.e., R_3 is modified) and T_1 (i.e., R_{17} and R_{20} are modified) need to be written back to the LLC after eviction from the tag cache. Conversely, only T_0 needs to be updated in the LLC in our approach, because all the replacement flags lie in T_0. Thus, our proposal would require to update a single tag column instead of updating two tag columns compared with LH-Cache for majority of cases. Since, the percentage of new CL insertions in the LLC is very low (less than 15% on average), the proposed policy reduces energy and bandwidth consumption by reducing the number of writebacks.

I. Tags-to-Column Mapping Policy

We rethink the design of the STT-RAM LLC to make it viable for a small RB organization discussed in Section II-D. For illustration, we assume that an STT-RAM bank is divided into four subarrays. Similarly, the large 2 KB STT-RAM row and RB are assumed to be partitioned into four small subrows and four sub-RBs, respectively, as shown in Fig. 13. Each subrow is stored in a different subarray. The primary advantage of a small RB organization is that multiple requests to different subarrays can be served in parallel. This is referred to as subarray parallelism [18]. If two accesses belong to different subrows of the same subarray, they must be accessed sequentially as each access will use the same subarray interface. This suppresses the subarray parallelism. To exploit the potential of subarray parallelism, it is pivotal to distribute requests to subarrays in an efficient manner. In the proposed tags-to-column mapping policy, the tag columns of adjacent cache sets/rows are stored in different subarrays as shown in Fig. 13(a). For instance, the tags of Set_0/Row_0, Set_1/Row_1, Set_2/Row_2, and Set_3/Row_3 are stored in $Subarray_0$, $Subarray_1$, $Subarray_2$, and $Subarray_3$, respectively. This way, it exploits the potential parallelism offered by independent subarrays within a bank. Furthermore, it does not incur any additional hardware cost as the least significant 2 bits of the row-id are used to identify the location of the tag columns. In our approach, simultaneous requests to access the tag columns from different subarrays can be accessed in parallel.

1) Comparison With LH-Cache: In contrast to our approach, LH-Cache always stores the tag columns in $Subarray_0$ as depicted in Fig. 13(b) which suppresses subarray parallelism. For performance comparison with the LH-Cache, let us consider an example of two LLC writeback accesses namely WB_1 and WB_2 as shown in Fig. 14. WB_1 performs a writeback to line L_{16} of Row_0, while WB_2 performs writeback to line L_{24} of Row_1. Note that both WB_1 and WB_2 bypass the RB and the tag cache as writebacks have limited spatial locality (refer to Section II-C). Fig. 14(a) shows the timeline of WB_1 and WB_2 being served by LH-Cache. This example highlights the problem of reduced subarray parallelism in LH-Cache. In LH-Cache, accesses to the tag columns of WB_1 and WB_2 are serialized. This significantly increases the overall service time of WB_1 and WB_2. Fig. 14(b) depicts the timeline of WB_1 and WB_2 being served by our approach. Since, the tag columns of WB_1 and WB_2 belong to different subarrays, accesses to WB_1 and WB_2 can be overlapped which significantly reduces the overall service time. Our approach uniformly distributes tag requests to different subarrays which allows overlapping a large number of requests compared with LH-Cache.

2) Comparison With LAMOST: Following a tag cache and an RB miss, an LLC read request requires to fetch the tag columns in the tag cache (recall Section III-D). An access to the tag columns in our approach requires a single subarray access, because the tag columns of a particular row reside in one subarray as shown in Fig. 13(a). In contrast, the LAMOST requires four subarray accesses for the tags access, because the four tag columns of a particular row resides in different subarrays as depicted in Fig. 13(c). Therefore, two concurrent tag column accesses in LAMOST cannot be overlapped even if they belong to different subrows of different subarrays of a particular bank. As a result, LAMOST significantly restrains subarray parallelism within a bank compared with our approach.

J. Overheads

The latency and energy values along with the overheads of various policies employing the 256 MB STT-RAM LLC are extracted using NVSIM [22], which are shown in Tables III and IV, respectively.

1) Selective Read Policy: The implementation of the selective read policy induces an additional hardware overhead. In order to keep track of which lines are currently present in the RB, we need to store a presence bit for each line in the row. This requires 28 bits per STT-RAM bank and a total of 224 bytes for our LLC organization with 64 banks. Besides the pure storage overhead, the selective read policy also requires two multiplexers (one for the read bypass and other for the write bypass as shown in Fig. 4). The total area overhead of the selective read policy is 0.28 mm^2. This includes the area required for the storage overhead and the multiplexers and the logic to enable/disable the sense amplifiers. The selective read policy marginally increases the access latency by one cycle (t_{EXT} in Table III) which is negligible compared with the higher access latency STT-RAM LLC. This is due to the fact that the additional multiplexers and demultiplexers required for our proposal require a little overhead compared with the larger STT-RAM bank (bank size is 4 MB for 64-bank STT-RAM LLC).

2) RB Tags-Bypass Policy: We take into an extra two-cycle latency for the RB tags-bypass policy which is required to transfer the tag columns from the bank array to the tag cache.

3) LLC Data Cache: The LDC requires an additional area overhead of 0.43 mm^2. The latency overhead is described in Section III-E (Fig. 11) and the energy overheads are enumerated in Table IV.

Overall, our proposal using 256-MB STT-RAM LLC requires an additional area overhead of 0.71 mm^2, which is an overhead of 0.2% compared with the area of the default STT-RAM configuration according to NVSIM (351 mm^2). Overall, the impact of our entire proposal (including selective read policy and the LDC) on area, energy, and latency is negligible.

IV. Evaluation

This section gives a brief overview of the simulator infrastructure that we used for evaluation and describes a set of benchmarks. It also presents qualitative and quantitative comparisons to state-of-the-art approaches.

TABLE II

SPEC2006 Application Mixes Used as Benchmarks for Evaluation. Values in Parenthesis Denote the Number of Instances Used for That Particular Application

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

TABLE III

Configuration Details as Used in the Experiments Along With Overheads

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
Shared LLC (DRAM or STT-RAM)	4 channels, 2 KB RB, 256 MB, 64 banks, 256-bit channel width, 2 cycle bus latency, t_{RCD}-t_{RP}-t_{CAS} = 18-18-18 (processor cycles)
t_{WR} t_{EXT} t_{HTree}	18 and 38 cycles for DRAM and STT-RAM respectively 1 cycle extra for *Selective Read Policy* 2 cycle extra for *RB-Tags-Bypass Policy*
Tag-Cache	27 KB, 2 cycle latency [6]–[8]
Miss Predictor	Map-I [16], 256 entries
Main Memory (DRAM)	2 channels, 16 KB row buffer, 64-bit channel width, 800 MHz bus frequency, tRAS-tRCD-tRP-tCAS-tWR = 144-36-36-36-36 (processor cycles)

A. Experimental Setup

We evaluated our STT-RAM LLC architecture using the Zesto X86 simulator [23]. Zesto provides detailed cycle accurate models of the core mircoarchitecture and of the cache hierarchy. For our benchmarks, we modeled an eight-core system where each core runs a single application from the SPEC2006 benchmark suite [19], [20]. We simulated a total of six application mixes as shown in Table II.

The parameters of the system configuration used for simulation are listed in Table III. We extensively modified the DDR memory model of Zesto to reflect the distinct characteristics of the STT-RAM similar to the work in [11] and [15]. Most importantly, we modeled the nonvolatility and high write latency of the STT-RAM. The modified STT-RAM LLC model considers bus contention, queuing delays, as well as bank and RB conflicts. We extracted the energy values of various microarchitectural structures using NVSIM [22] and Cacti [24], [25] and configured Zesto accordingly. Table IV shows the energy values for various operations along with overheads. These values show the energy consumed by row and column decoders, sense amplifiers, multiplexers, write drivers, and read latches.

For evaluation, we compared the following LLC variants.

1) *DRAM-Base:* DRAM LLC without any optimizations [7], [10].
2) *STT-Base:* STT-RAM LLC without any optimizations.

TABLE IV
DYNAMIC ENERGY CONSUMPTION OF VARIOUS OPERATIONS

Operation	Energy Baseline	Energy with overheads
DRAM Array Read/Write[1]	14.13	NA[3]
DRAM Precharge[1]	4.63	NA[3]
DRAM RB Access[1]	11.88	NA[3]
STT-RAM Array Read[1]	10.87	11.54
STT-RAM Array Write[1]	28.49	29.32
STT-RAM RB Access[1]	10.07	10.72
Tag-Cache Access (27.2 KB)[2]	0.97	0.97
LLC Data Cache (LDC) (34.5 KB)[2]	NA	1.12

[1] nJ/Column; [2] nJ/Access; item[3] NA: Not applicable

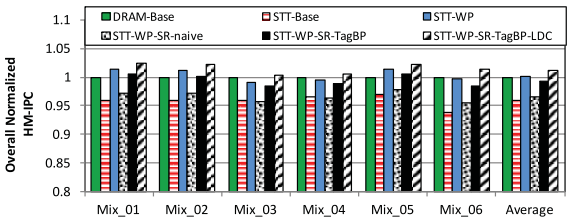

Fig. 15. Measured harmonic mean instruction per cycle (HM-IPC) for the six application mixes and different configurations.

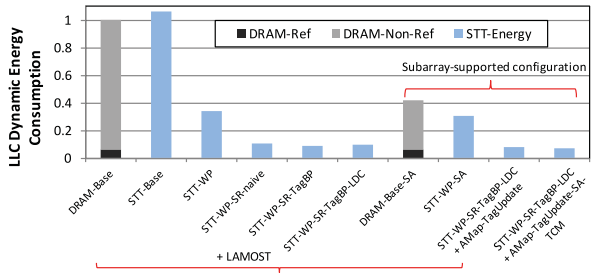

Fig. 16. Average LLC dynamic energy consumption for various configurations normalized to DRAM-base configuration [7], [10]. Refer to Section IV-D for subarray parallelism results.

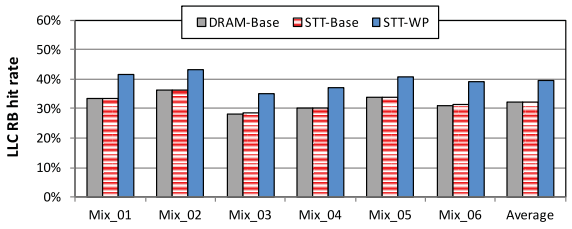

Fig. 17. LLC RB hit rate for various configurations.

3) *STT-WP:* STT-RAM LLC including the state-of-the-art writeback RB bypass [11] and partial write optimizations [15] described in Section II-C.
4) *STT-WP-SR-naive:* State-of-the-art STT-RAM LLC extended by our selective read policy as described in Section III-B using existing tag read policy in [7] and [8].
5) *STT-WP-SR-TagBP:* State-of-the-art STT-RAM LLC extended by our selective read policy using the RB Tags-Bypass Policy from Section III-D.
6) *STT-WP-SR-TagBP-LDC:* STT-WP-SR-TagBP extended by the LDC from Section III-E.

For all configurations, we make the following assumptions.

1) We use the LAMOST LLC set mapping policy [7], [10] (see Fig. 2)
2) We assume an LLC with four channels, 256-bits bus width per channel, two cycles bus latency, and 2-KB RB size.
3) We assume that writes to the STT-RAM array takes 20 cycles more compared with a DRAM array. This implies that t_{WR} (write recovery time) is 18 cycles for the DRAM and 38 cycles for the STT-RAM.
4) We employ the Simpoint tool [26] to choose the region of interest for each benchmark.
5) We use first ready first come first serve access scheduling [27] for the LLC.

B. Performance and Energy Measurements

Fig. 15 shows the simulation results for all described configurations normalized to DRAM-Base configuration. As depicted, our STT-WP-SR-TagBP-LDC configuration improves the average performance by 1.2%, 5.4%, and 1.1% compared with DRAM-Base, STT-Base, and STT-WP configurations, respectively.

The energy benefits of the STT-WP-SR-TagBP-LDC configuration can be observed in Fig. 16. As shown, the average energy consumption is reduced by 89.7% compared with DRAM-Base and 90.5% compared with STT-Base. While the writeback RB bypass policy proposed in [11] already saves a large amount of energy, our selective read policy further reduces this energy consumption by 69.4% by eliminating unnecessary STT-RAM reads.

In the following, we compare the various configurations in more detail.

1) Comparing DRAM-Base and STT-Base: Simply replacing the DRAM arrays by STT-RAM arrays does not provide any benefits in performance or energy consumption. On the contrary, we observe an average performance degradation of 4.3% and a 6.1% increased energy consumption for STT-RAM compared with the DRAM. This is mostly due to the high latency and energy consumption of write operations in the STT-RAM. However, due to the decoupled organization of the STT-RAM, our optimizations can reduce the number of write operations in an STT-RAM LLC and significantly decrease the energy consumption. This is illustrated in the following.

2) Comparing STT-Base and STT-WP: Applying the writeback RB bypass policy [11] significantly improves the performance and energy consumption of an STT-RAM LLC. In STT-Base configuration, a writeback always overwrites the RB, which might cause eviction of highly reused rows. The writeback row, however, is unlikely to be reused on a subsequent access. Bypassing the RB for writeback operations increases the RB hit rate from 32.2% to 39.5% as illustrated in Fig. 17. This improves the performance by 4.8% on average for our application mixes.

In addition to the writeback RB bypass policy, the STT-WP configuration also applies the partial write optimization [15]. As a consequence of both optimizations, the LLC dynamic energy consumption is reduced by 69.2% compared with the STT-Base configuration.

3) Impact of the Selective Read Policy: Although the STT-WP configuration is more energy-efficient than the

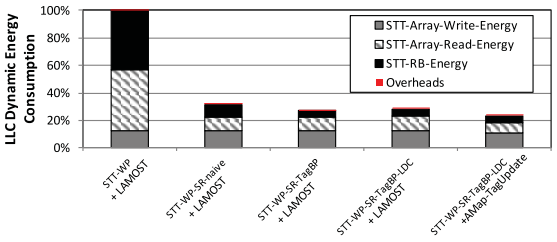

Fig. 18. Average LLC dynamic energy consumption of various configurations. Values are normalized to the STT-WP configuration for reference. Overheads include the energy consumed by the tag cache and the LDC.

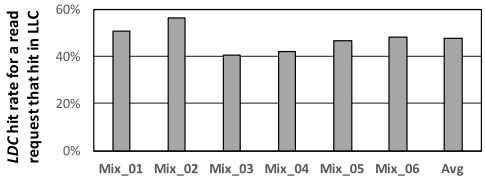

Fig. 19. LDC hit rate for a read request that hits in LLC.

STT-Base configuration, it still incurs an unnecessarily high energy consumption by reading unneeded lines from the STT-RAM array to the RB. Applying our selective read policy (see Section III-B) further reduces the energy consumption by 69.4% compared with the STT-WP configuration. Fig. 18 illustrates the energy improvement of configurations that use the selective read policy. However, the selective read policy degrades the performance by an average of 3.7% compared with the STT-WP configuration (see Fig. 15) due to an increased tag-cache/RB miss latency [see Fig. 8(a)].

4) Impact of the RB Tags-Bypass Policy: Since the selective read policy suffers from an increased tag-cache/RB miss latency, it has a limited performance. To overcome this limitation, we proposed the RB tags-bypass policy (Section III-D), which bypasses the RB for the tag access. As shown in Fig. 8, this optimization reduces the latency from 86 to 70 cycles. Furthermore, the RB tags-bypass policy reduces the energy consumption, as it avoids duplicate storage of tags. As a result, the STT-WP-SR-TagBP configuration improves the average performance by 2.8% and reduces the LLC dynamic energy consumption of 14.5% compared with the STT-WP-SR-naive configuration.

5) Impact of the LDC: The STT-WP-SR-TagBP configuration improves performance compared with the STT-WP-SR-naive configuration. However, it still degrades the performance by 0.9% on average compared with the state-of-the-art STT-WP configuration. To further improve the performance, we proposed a small structure called LDC (see Section III-E). The inclusion of LDC improves the performance by 1.1% and 2.1% compared with the STT-WP and STT-WR-SR-TagBP configurations. The proposed LDC accelerates access latency to some CLs as an STT-RAM read can be avoided when the line is present in the LDC. As illustrated in Fig 19, the LDC hit rate for a read request is 47.7% on average for all application mixes. On the downside, the STT-WP-SR-TagBP-LDC configuration increases LLC dynamic energy consumption by 4.4% compared with the STT-WP-SR-TagBP configuration due to reading extra columns from the STT-RAM array.

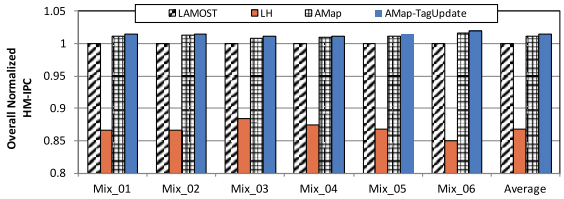

Fig. 20. Harmonic mean instruction per cycle for different evaluated configurations.

C. Impact of Address Organization and Tags-Update Policy

This section provides detailed qualitative and quantitative comparisons of our address mapping and tags-update policy when applied to STT-WP-SR-TagBP-LDC configuration. We select this case, since it achieves the best performance results compared with other configurations (Fig. 15). We evaluate the following new LLC configurations.

1) *LAMOST:* State-of-the-art address mapping namely LAMOST [10] in Section II-A (Figs. 2 and 3) applied to STT-WP-SR-TagBP-LDC configuration.
2) *LH:* STT-WP-SR-TagBP configuration extended with LH-Cache [9], [12] address mapping [see left column of Fig. 12(d) and (e)]. This configuration does not support LDC, because filling the LDC would require an access to a different row which is not a viable option in LH-Cache.
3) *AMap:* The STT-WP-SR-TagBP-LDC configuration extended with our tag organization and address mapping as discussed in Sections III-F and III-G, respectively [see right column of Fig. 12(d) and (e)].
4) *AMap-TagUpdate:* The AMap configuration extended with our tag update policy in Section III-H.

As shown in Fig. 20, our AMap-TagUpdate configuration improves the HM-IPC throughput by 1.4% and 16.6% compared with LAMOST and LH configurations, respectively. By modifying the way the tags and replacement flags are stored in the tag columns (see Section III-H), the AMap-TagUpdate configuration reduces the number of tag column updates by 23% compared with the AMap configuration. As a result, the AMap-TagUpdate configuration improves the performance by 0.3% via reducing the LLC bandwidth. The performance of the STT-RAM LLC strongly relies on the RB hit rate (higher is better), the tag-cache hit rate (higher is better), the LLC miss rate (lower is better; depends upon associativity), the tag-cache/RB miss latency (lower is better), and the LDC hit rate (higher is better). Table V provides a quantitative comparison of these parameters for the evaluated configurations. The performance enhancement using our synergistic policies is obtained via improvement of all important parameters as presented in Table V.

The LH and the variants of AMap configurations (i.e., AMap and AMap-TagUpdate) have the benefits of high associativity [i.e., 30 way; Fig. 12(a)] compared with the LAMOST configuration [i.e., seven way; Fig. 2(a)] which reduces an average LLC miss rate. In addition, these configurations have a low tag-cache/RB miss latency (i.e., 66 cycles) compared with the LAMOST configuration (i.e., 70 cycles). However, the benefits of LH configuration comes at the cost of a significantly low RB and tag-cache hit rates (see Table V)

TABLE V
COMPARISONS OF DIFFERENT CONFIGURATIONS. THE RED COLOR INDICATES A BAD VALUE FOR A PARAMETER

Configuration	Demand Tag-Cache Hit Rate	Avg. RB Hit Rate	Avg. LLC Miss Rate	Tag-Cache/RB Miss Latency *Selective Read policy*	Demand *LDC* Hit Rate	Subarray Parallelism
LH [9], [12]	**15.2%**	**6.35 %**	20.2%	66 (Fig. 8c)	Not applicable	**Bad**
LAMOST [10]	61.8%	39.6%	**23.2%**	**70 (Fig. 8b)**	47.7%	**Bad**
AMap-TagUpdate[Ours]	61.5%	39.5%	21.4%	66 (Fig. 8c)	47.5%	Good

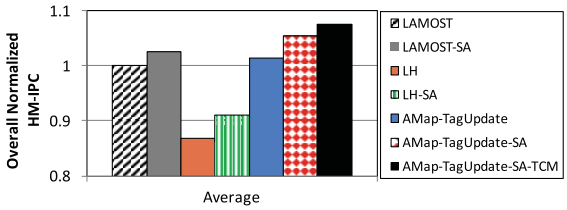

Fig. 21. Performance results showing the impact of subarray parallelism.

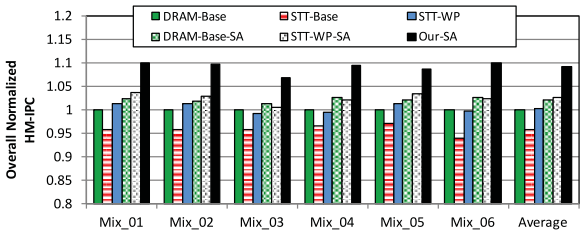

Fig. 22. Performance results showing the impact of our synergistic policies.

due to reduced spatial and temporal locality. The proposed AMap configurations benefit from the high associativity and low tag-cache/RB miss latency benefits of LH configuration while covering its shortcomings by changing the address mapping policy (see Section III-G). Both AMap and LAMOST configurations map four consecutive memory blocks to the same LLC row which result in almost similar RB, tag-cache, and LDC hit rates as illustrated in Table V. In comparison with the LAMOST configuration, the AMap-TagUpdate configuration reduces the STT-RAM energy consumption by 17.1% via reading and writing less number of tag columns as depicted in Fig. 18.

D. Impact of Subarray Parallelism and Tags-to-Column Mapping Policy

This section evaluates the performance improvement of subarray parallelism (Section II-D) on an LLC organization with four subarrays per bank. For evaluation, we consider the following configurations.

1) *LAMOST-SA & LH-SA:* The LAMOST and LH configurations extended with subarray parallelism, respectively.
2) *AMap-TagUpdate-SA:* AMap-TagUpdate extended with subarray parallelism.
3) *AMap-TagUpdate-SA-TCM:* AMap-TagUpdate-SA configuration extended with our tags-to-column mapping policy in Section III-H.

Fig. 21 demonstrates the performance benefits of subarray-parallelism extended configurations over subarray-oblivious configurations. We make three key observations. First, the subarray-supported configurations outperform their corresponding subarray-oblivious configurations. Second, the LAMOST-SA and LH-SA configurations do not get noticeable performance benefits (i.e., 2.3% and 4.8%, respectively) compared with their corresponding subarray-oblivious (i.e., LAMOST and LH) configurations. Third, the performance impact of our tags-to-column mapping policy is more significant compared with the LAMOST-SA and LH-SA configurations. Our AMap-TagUpdate-SA-TCM configuration outperforms the subarray-oblivious AMap-TagUpdate configuration by 6.5%.

Based on these observations, we conclude that the incorporation of tags-to-column mappying policy in our subarray supported proposal allows overlapping of different requests and improves the overall performance. While in other configurations, it is not as effective. In the LAMOST-SA configuration, all four tag columns are located in four different subarrays of a bank. This requires accessing all subarrays for reading the tag columns, which suppresses subarray parallelism. Similarly, in the LH-SA configuration, $Subarray_0$ is overutilized, because it stores the tag columns which are accessed after a tag-cache miss before accessing the cache line in other subarrays. Other subarrays ($Subarray_1$, $Subarray_2$, and $Subarray_3$) store the cache lines and are less frequently accessed.

E. Putting It All Together

This section provides the performance and energy gains of our synergistic policies relative to the configurations described in Section IV-A. Our-SA configuration combines the performance advantages of RB tags-bypass policy, LDC, address organization, tags-update policy, tags-to-column mapping policy, and subarray parallelism. The net performance gain is substantially higher than the performance gain of a single policy. On average, Our-SA configuration improves the HM-IPC throughput by 8.9%, 13.6%, 8.7%, 6.8%, and 6.5% compared with DRAM-Base, STT-Base, STT-WP, DRAM-Base-SA, and STT-WP-SA configurations, respectively, as shown in Fig. 22.

In addition to performance improvement, the proposed configuration substantially reduces the dynamic energy consumption. It reduces the energy consumption by 92%, 92.4%, 78%, 82%, and 75%relative to DRAM-Base, STT-Base, STT-WP, DRAM-Base-SA, and STT-WP-SA configurations as depicted in Fig. 16. This energy reduction is brought by the combination of selective read policy, address organization, and the tags-update policy. The selective read policy is the leading contributor in this energy reduction (the exact breakdown is discussed in Sections IV-B–IV-D).

V. RELATED WORK

Existing LLC architectures (mostly DRAM-based) can be categorized into block-based and page-based designs.

Block-based LLCs [6]–[10], [16], [28]–[30] use a small line size (i.e., 64 byte), whereas page-based LLCs [31]–[35] use a large line size (i.e., 1 KB/2 KB). The advantage of page-based designs is that they leverage the spatial locality of applications by storing entire pages in the LLC. However, the large page size has a high price. The excessive prefetching leads to a high usage of memory bandwidth which makes it unsuitable for multicore system running diverse applications. Another drawback of the page-based LLC is that not all memory blocks within the large page are accessed prior to page evictions, which may exacerbate the performance. In contrast to page-based LLCs, we employ a block-based LLC to mitigate the excessive memory bandwidth utilization problem.

The large memory requirements of new applications have forced the industry to increase the LLC size. For instance, the IBM POWER7 processor [36], [37] uses a 32-MB DRAM LLC. The DRAM-based LLC architectures [6]–[10], [16], [28]–[35] consume a significant portion of chip power budget due to high leakage and refresh energy which increases with the LLC size. To mitigate the power scalability of the DRAM, recently STT-RAM LLC architecture [11] has been proposed as a promising alternative to the DRAM LLC.

Independent of the LLC architecture (i.e., block-based or page-based; the DRAM-based or STT-RAM-based), all of the above-mentioned studies always fetch the content of an entire row into the RB after an RB miss. This leads to an unnecessarily high energy consumption, since most of the columns will not be used. Our proposal is unique in the sense that it only fetches those lines into the RB which are likely to be accessed later.

Many architectural techniques have been presented for energy and performance tradeoffs in the STT-RAM caches [38]–[40]. They relax the nonvolatility of the STT-RAM by tuning the MTJ volume to improve its write latency and energy requirements at the cost of additional refresh overheads. However, these techniques are not suitable for a larger STT-RAM due to high energy requirements for refreshing MTJ cells periodically. In contrast, our proposal exploits the nonvolatility characteristics of the STT-RAM.

Other circuit-level energy and latency reduction techniques that exploit heterogeneity in the switching time of the STT-RAM bit cell have been introduced in [41] and [42]. However, these circuit-level techniques are orthogonal to this paper and can be combined with our proposal in order to further improve performance and energy efficiency of the STT-RAM LLC.

It is worth to mention that the concepts proposed in this paper are generic and can be applied to other NVM memory technologies, such as PCM and ReRAM subject to the condition that the latency of NVM-based LLC is significantly less compared with the off-chip memory.

VI. Conclusion

This paper presents novel policies to improve the performance and energy efficiency of STT-RAM LLC architectures. We demonstrate that existing LLC architectures fetch large amounts of data into the RB while the majority of the data are not reused. Our selective read policy exploits this fact and reduces the energy consumption by decreasing the number of STT-RAM LLC reads. To mitigate the additional latency incurred by the policy, we propose a new RB tags-bypass policy and an LDC organization. For energy reduction, we propose a novel tags-update policy that updates less number of tag columns compared with the existing approaches. Our tags-to-column mapping policy exploits the inherent potential offered by subarray parallelism by uniformly distributing requests to different subarrays. Our results on SPEC 2006 benchmarks show that our synergistic policies are effective in improving the average performance (8.9% and 6.5%) and energy consumption (82% and 75%) compared with the state state-of-the-art approaches for the DRAM and the STT-RAM LLC. With these optimizations, the STT-RAM becomes an effective LLC alternative to the DRAM.

References

[1] D. Gove, "CPU2006 working set size," *SIGARCH Comput. Archit. News*, vol. 35, no. 1, pp. 90–96, Mar. 2007.

[2] W. A. Wulf and S. A. McKee, "Hitting the memory wall: Implications of the obvious," *ACM SIGARCH Comput. Archit. News*, vol. 23, no. 1, pp. 20–24, Mar. 1995.

[3] C. Weis, M. Jung, and N. Wehn, "3D memories," *Handbook of 3D Integration*, vol. 4. Weinheim, Germany: Wiley-VCH, 2016.

[4] (2013). *Hybrid Memory Cube Consortium: Hybrid Memory Cube Specification*. http://www.jedec.org/standards-documents/docs/jesd235

[5] U. Kang *et al.*, "8 Gb 3-D DDR3 DRAM using through-silicon-via technology," *IEEE J. Solid-State Circuits*, vol. 45, no. 1, pp. 111–119, Jan. 2010.

[6] C.-C. Huang and V. Nagarajan, "ATCache: Reducing DRAM cache latency via a small SRAM tag cache," in *Proc. 23rd Int. Conf. Parallel Architect. Compilation Techn. (PACT)*, Aug. 2014, pp. 51–60.

[7] F. Hameed, L. Bauer, and J. Henkel, "Reducing latency in an SRAM/DRAM cache hierarchy via a novel tag-cache architecture," in *Proc. 51st Design Autom. Conf. (DAC)*, Jun. 2014, pp. 1–6.

[8] F. Hameed, L. Bauer, and J. Henkel, "Architecting on-chip DRAM cache for simultaneous miss rate and latency reduction," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 4, pp. 651–664, Apr. 2016.

[9] G. Loh and M. Hill, "Supporting very large DRAM caches with compound-access scheduling and MissMap," *IEEE Micro*, vol. 32, no. 3, pp. 70–78, May/Jun. 2012.

[10] F. Hameed, L. Bauer, and J. Henkel, "Simultaneously optimizing DRAM cache hit latency and miss rate via novel set mapping policies," in *Proc. Int. Conf. Compilation, Archit., Synth. Embedded Syst. (CASES)*, Sep./Oct. 2013, pp. 1–10.

[11] F. Hameed and M. B. Tahoori, "Architecting STT last-level-cache for performance and energy improvement," in *Proc. 17th Int. Symp. Quality Electron. Design (ISQED)*, Mar. 2016, pp. 319–324.

[12] G. H. Loh and M. D. Hill, "Efficiently enabling conventional block sizes for very large die-stacked DRAM caches," in *Proc. 44th IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Dec. 2011, pp. 454–464.

[13] F. Hameed and J. Castrillon, "Rethinking on-chip DRAM cache for simultaneous performance and energy optimization," in *Proc. 19th Conf. Design, Autom. Test Eur. (DATE)*, Mar. 2017, pp. 362–367.

[14] F. Hameed, C. Menard, and J. Castrillon, "Efficient STT-RAM last-level-cache architecture to replace DRAM cache," in *Proc. Int. Symp. Memory Syst. (MemSys)*, Oct. 2017, pp. 141–151.

[15] E. Kültürsay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu, "Evaluating STT-RAM as an energy-efficient main memory alternative," in *Proc. Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Apr. 2013, pp. 256–267.

[16] M. K. Qureshi and G. H. Loh, "Fundamental latency trade-off in architecting DRAM caches: Outperforming impractical SRAM-tags with a simple and practical design," in *Proc. 45th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Dec. 2012, pp. 235–246.

[17] J. Meza, J. Li, and O. Mutlu, "A case for small row buffers in non-volatile main memories," in *Proc. IEEE 30th Int. Conf. Comput. Design (ICCD)*, Sep. 2012, pp. 484–485.

[18] Y. Kim, V. Seshadri, D. Lee, J. Liu, O. Mutlu, "A case for exploiting subarray-level parallelism (SALP) in DRAM," in *Proc. 39th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2012, pp. 368–379.

[19] (2017). *Standard Performance Evaluation Corporation*. Accessed: Mar. 10, 2017. [Online]. Available: http://www.spec.org
[20] J. L. Henning, "SPEC CPU2006 benchmark descriptions," *ACM SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 1–17, Sep. 2006.
[21] J. Feehrer *et al.*, "The oracle sparc T5 16-core processor scales to eight sockets," *IEEE Micro*, vol. 33, no. 2, pp. 48–57, Mar./Apr. 2013.
[22] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "NVSim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 31, no. 7, pp. 994–1007, Jul. 2012.
[23] G. H. Loh, S. Subramaniam, and Y. Xie, "Zesto: A cycle-level simulator for highly detailed microarchitecture exploration," in *Proc. Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Apr. 2009, pp. 53–64.
[24] S. Thoziyoor, J. Muralimanohart, R. Ahn, and N. Jouppi, "CACTI 5.1," HP Labs, Palo Alto, CA, USA, Tech. Rep. HPL-2008-20, Apr. 2008.
[25] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0," in *Proc. 40th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Dec. 2007, pp. 3–14.
[26] G. Hamerly, E. Perelman, J. Lau, and B. Calder, "Simpoint 3.0: Faster and more flexible program analysis," *J. Instruct. Level Parallelism*, vol. 7, pp. 1–28, Sep. 2005.
[27] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, "Memory access scheduling," in *Proc. IEEE Comput. Soc. 27th Annu. Int. Symp. Comput. Archit.*, Vancouver, BC, Canada, Jun. 2000, pp. 128–138.
[28] G. H. Loh, "Extending the effectiveness of 3D-stacked DRAM caches with an adaptive multi-queue policy," in *Proc. 42nd Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, 2009, pp. 174–183.
[29] J. Sim, G. H. Loh, H. Kim, M. O'Connor, and M. Thottethodi, "A mostly-clean DRAM cache for effective hit speculation and self-balancing dispatch," in *Proc. 45th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Dec. 2012, pp. 247–257.
[30] J. Sim, G. H. Loh, V. Sridharan, and M. O'Connor, "Resilient die-stacked DRAM caches," in *Proc. 40th Int. Symp. Comput. Architect. (ISCA)*, 2013, pp. 416–427.
[31] D. H. Woo, N. H. Seong, D. L. Lewis, and H.-H. S. Lee, "An optimized 3D-stacked memory architecture by exploiting excessive, high-density TSV bandwidth," in *Proc. 16th IEEE Symp. High-Perform. Comput. Archit. (HPCA)*, Jan. 2010, pp. 1–12.
[32] X. Jiang *et al.*, "CHOP: Adaptive filter-based DRAM caching for CMP server platforms," in *Proc. 16th IEEE Symp. High-Perform. Comput. Archit. (HPCA)*, Jan. 2010, pp. 1–12.
[33] X. Jiang *et al.*, "CHOP: Integrating DRAM caches for CMP server platforms," *IEEE Micro*, vol. 31, no. 1, pp. 99–108, Jan./Feb. 2011.
[34] 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 *Proc. 40th Int. Symp. Comput. Archit. (ISCA)*, 2013, pp. 404–415.
[35] D. Jevdjic, G. H. Loh, C. Kaynak, and B. Falsafi, "Unison cache: A scalable and effective die-stacked dram cache," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchitect. (MICRO)*, Dec. 2014, pp. 25–37.
[36] D. Wendel, "The implementation of POWER7: A highly parallel and scalable multi-core high-end server processor," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2010, pp. 102–103.
[37] R. X. Arroyo, R. J. Harrington, S. P. Hartman, and T. Nguyen, "IBM POWER7 systems," *IBM J. Res. Develop.*, vol. 55, no. 3, pp. 2:1–2:13, 2011.
[38] A. Jog *et al.*, "Cache revive: Architecting volatile STT-RAM caches for enhanced performance in CMPs," in *Proc. 49th IEEE/ACM Design Autom. Conf. (DAC)*, Jun. 2012, pp. 243–252.
[39] C. W. Smullen, V. Mohan, A. Nigam, S. Gurumurthi, and M. R. Stan, "Relaxing non-volatility for fast and energy-efficient STT-RAM caches," in *Proc. IEEE 17th Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2011, pp. 50–61.
[40] Z. Sun *et al.*, "Multi retention level STT-RAM cache designs with a dynamic refresh scheme," in *Proc. 44th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Dec. 2011, pp. 329–338.
[41] R. Bishnoi, F. Oboril, M. Ebrahimi, and M. Tahoori, "Avoiding unnecessary write operations in STT-MRAM for low power implementation," in *Proc. 15th Int. Symp. Quality Electron. Design (ISQED)*, Mar. 2014, pp. 548–553.
[42] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "Energy reduction for STT-RAM using early write termination," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2009, pp. 264–268.

Fazal Hameed received the 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, Technische Universität Dresden, Dresden, Germany, as a Postdoctoral Researcher in 2016. He held a similar position at the Chair of Dependable and Nano Computing, KIT. He is also with the Institute of Space Technology, Islamabad, Pakistan. He mainly involved in the architecture group with a focus on memories. He is currently involved in the development of a simulation framework to evaluate the performance, energy, and reliability of heterogenous multicore system architecture.

Dr. 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.

Asif Ali Khan received the B.S. and M.S. degrees in computer systems engineering from the University of Engineering and Technology, Peshawar, Pakistan, in 2012 and 2015, respectively. He is working toward the Ph.D. degree at the Chair for Compiler Construction, Computer Science Department, Technische Universität Dresden, Dresden, Germany.

His current research interests include computer architecture, heterogeneous memories, and compiler support for memory systems.

Jeronimo Castrillon received the bachelor's degree in electronics engineering from Pontificia Bolivariana University, Medellín, Colombia, in 2004, the M.S. degree from the Advanced Learning and Research Institute, Lugano, Switzerland, in 2006, and the Ph.D. (Dr.-Ing.) degree (honors) from the RWTH Aachen University, Aachen, Germany, in 2013.

He is currently a Professor with the Department of Computer Science, Technische Universität Dresden, where he is with the Center for Advancing Electronics Dresden. He is also the Head of the Chair for Compiler Construction, with research focus on methodologies, languages, tools, and algorithms for programming complex computing systems.

Dr. Castrillon is a member of the Executive Committee of the ACM Future of Computing Academy since 2017.