Efficient STT-RAM Last-Level-Cache Architecture to replace DRAM Cache

Fazal Hameed
fazal.hameed@tu-dresden.de
Technische Universität Dresden
Center for Advancing Electronics Dresden
Germany

Christian Menard
christian.menard@tu-dresden.de
Technische Universität Dresden
Center for Advancing Electronics Dresden
Germany

Jeronimo Castrillon
jeronimo.castrillon@tu-dresden.de
Technische Universität Dresden
Center for Advancing Electronics Dresden
Germany

ABSTRACT

Recent research has proposed die-stacked Last Level Cache (LLC) to overcome the Memory Wall. Lately, Spin-Transfer-Torque Random Access Memory (STT-RAM) caches have been recommended as they provide improved energy efficiency compared to DRAM caches. However, the recently proposed STT-RAM cache architecture unnecessarily dissipates energy by fetching unneeded cache lines into the row buffer. In this paper, we propose a *Selective Read Policy* for STT-RAM. This policy only fetches those cache lines into the row buffer that are likely to be reused. This reduces the number of cache line reads and thereby reduces the energy consumption. Further, we propose two key performance optimizations namely *Row Buffer Tags Bypass Policy* and *LLC Data Cache*. Both optimizations reduce the LLC access latency and therefore improve the overall performance. For evaluation, we implement our proposed architecture in the Zesto simulator and run different combinations of SPEC2006 benchmarks on an 8-core system. We show that our synergetic policies reduce the average LLC dynamic energy consumption by 72.6% and improve the system performance by 1.3% compared to the recently proposed STT-RAM LLC. Compared to the state-of-the-art DRAM LLC, our architecture reduces the LLC dynamic energy consumption by 90.6% and improves system performance by 1.4%.

CCS CONCEPTS

•Hardware → Non-volatile memory;

ACM Reference format:
Fazal Hameed, Christian Menard, and Jeronimo Castrillon. 2017. Efficient STT-RAM Last-Level-Cache Architecture to replace DRAM Cache. In *Proceedings of International Symposium on Memory Systems, Washington DC, USA, October 2–5, 2017 (MEMSYS'17),* 11 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION AND MOTIVATION

Conventional off-chip memories do not fulfil the bandwidth and latency requirements of complex applications running on multi-core systems [6]. 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 [41] 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 [1, 22, 24, 38]. These technologies integrate processor and memory dies employing a low latency and high bandwidth interconnect. To abate 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) [7–13, 16, 27].

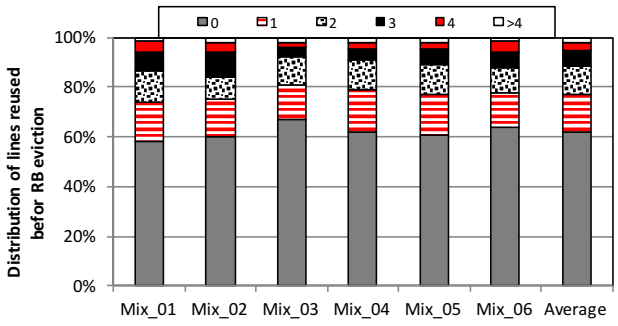


Figure 1: Distribution of number of unique lines reused before RB eviction using 2KB RB size for SPEC2006 application mixes (see Table 2)

Previously, on-chip DRAM memory has been adopted as LLC for performance improvement due to its capacity advantage compared to an area equivalent SRAM cache [9–11, 16, 27]. However, the DRAM LLC dissipates a significant portion of chip power budget due to its high refresh and dynamic energy consumption. Therefore, recent research has advocated the use of non-volatile Spin-Transfer-Torque Random Access Memory (STT-RAM) as LLC [13]. By exploiting the non-volatility characteristics of STT-RAM, the energy consumption of the LLC can be reduced significantly compared to DRAM.

Typically, a high capacity STT-RAM holds multiple banks and each bank is provided with a Row Buffer (RB) [13, 23]. The total energy consumption in existing STT-RAM LLC is dominated by reading data from an STT-RAM bank into the RB [13]. To reduce the energy consumption of STT-RAM LLC, we exploit the fact that most of the cache lines are unnecessarily fetched into the RB as they are not likely to be reused in the near future. Figure 1 shows

MEMSYS'17, Washington DC, USA

DOI: 10.1145/nnnnnnn.nnnnnnn

that the probability for the RB content to be accessed before RB eviction is less than 40%. Therefore, we fetch only those cache lines into the RB that are likely to be reused. This reduces the number of RB fetches significantly and, therefore, also reduces the dynamic energy consumption of an STT-RAM LLC. However, identifying the likelihood for a cache line to be reused induces a high tag access latency when using existing tag read policy [9, 13, 27]. Therefore, we also propose a novel *RB Tags Bypass Policy* that provides fast access to LLC tags. More precisely, we make the following contributions:

(1) We classify lines of an 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. By avoiding the need to fetch *lowly-reused* lines, we reduce the energy consumption significantly.
(3) While the *Selective Read Policy* saves energy it also induces a latency penalty. To improve the performance, we propose a *RB Tags Bypass Policy*.
(4) To further improve the performance, we also propose a small *LLC Data Cache* (LDC) that stores lines which are likely to be accessed later. Lines that hit in the *LDC* are accessed with much lower latency.

2 BACKGROUND

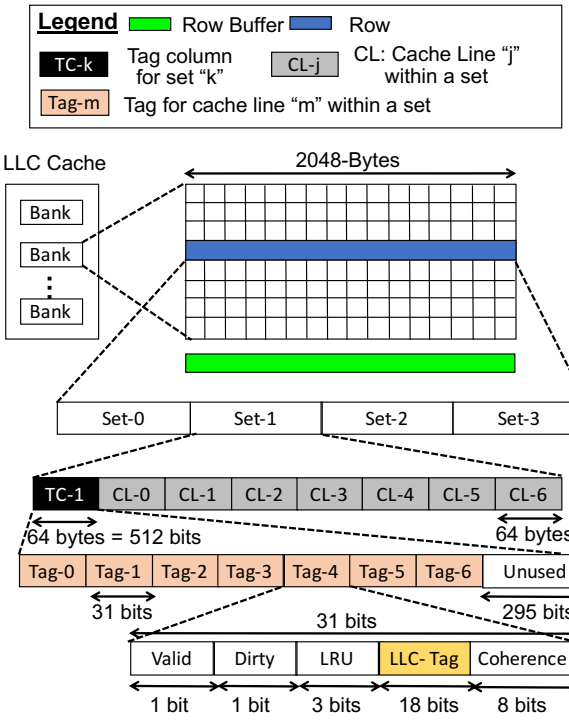
This section presents details of the organization for the recently proposed LLC. Further, this section describes the basic operating principles of STT-RAM and qualitatively compares STT-RAM to DRAM.

2.1 State-of-the-art LLC Organization

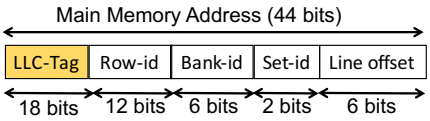
Figure 2a illustrates the organization of the previously proposed LLC [9, 10, 13]. The LLC is split into multiple memory banks each equipped with a row buffer (RB). Each bank is organized as a series of rows that are split into columns. In this paper, we assume an LLC size of 256 MB, a row size of 2048 bytes and a column size of 64 bytes.

The LLC is implemented as a set-associative cache. Recent research proposed various set mapping policies [9, 10, 13, 27, 31]. For this paper, we use the most recently proposed policy *LAMOST* that was used for DRAM [9, 10] LLCs and STT-RAM LLC [13]. In the LAMOST policy, the tags and lines of an LLC are stored in the same row. Each 2KB LLC row comprises 4 sets with 7-way associativity. Each set contains 7 cache lines and a tag column. The cache lines have a size of 64 bytes. Each tag has a size of 31 bits and in each tag column $7 \times 31 = 217$ bits are used for the 7 tag entries. This leaves 295 bits of each tag column unused.

In this paper, we assume 44-bit physical addresses. The address is split into multiple parts as depicted in Figure 2b. The 18 most significant bits identify the LLC-tag. The next 12 bits select a row within a bank and the following 6 bits select a memory bank within the LLC. 2 bits are used to identify the set within a row and the remaining 6 least significant bits identify a byte within a cache line. Figure 3 shows how main memory blocks are mapped to a particular bank, row, and set using the LAMOST policy. All the



(a) LLC LAMOST organization



(b) Address organization

Figure 2: Typical LLC and address organization using the LAMOST policy [9, 10, 13] for an 8-core system with 256 MB LLC size, 64 banks, 2 kB row buffer size and 44-bit physical addresses

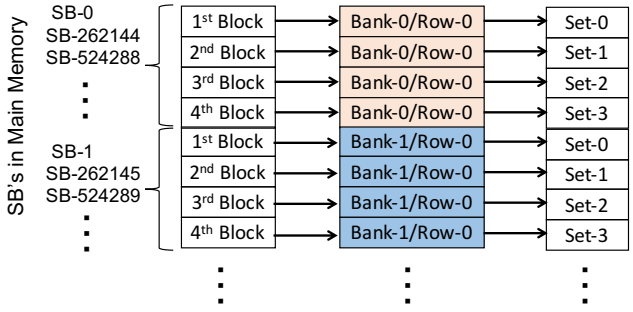


Figure 3: Bank/row/set mapping for *LAMOST* policy [9, 10, 13]. SB stands for super-block

blocks that are mapped to the same row are referred to as a super-block (SB). With the LAMOST policy, each super-block consists of 4 consecutive 64 byte memory blocks.

For each bank, the RB holds a copy of the row that was retrieved last. The RB allows for exploiting bank-level parallelism while keeping a shared command, address, and data bus. A controller or access scheduler is employed do avoid any conflicts on the shared buses. In order to access a cache line, the controller issues a command that fetches the corresponding row from the bank to the bank's row buffer. Successive requests to cache lines that are contained within the same row are serviced directly without fetching the row again. This is referred to as a row buffer hit. An access to a line that is contained in a different row, requires to fetch this row into the RB. This is referred to as a row buffer miss. The access latency and energy of an RB hit is much lower compared to that of an RB miss. Therefore, an application with high row buffer locality (many row buffer hits) has better performance and lower energy consumption than a similar application with low row buffer locality.

While we assume a specific setting with fixed row, column, LLC, and physical address size, the concepts presented in this paper can be applied to other settings as well.

2.2 Basic Operating Principles of DRAM and STT-RAM

In DRAM, data is 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'. The sense amplifier consists of cross-coupled transistors that amplify the small differential signal that appears on the bitline following the charge transfer from the bit-cell. In DRAM, all the sense amplifiers that are connected to the bitlines are referred to as the row buffer. 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* (see Figure 4a). 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 anti-parallel 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 anti-parallel state, the resistance is high and the cell stores a logical '1'. To read the value from a bit-cell, a voltage is applied between the source line and the bitline (see Figure 4b). This causes a low current to flow through the MTJ. The sense amplifier uses this current to detect the resistance state of the MTJ cell and decides for logical '0' or logical '1', which then can be stored in the row buffer.

2.3 STT-RAM RB Bypass and Partial Write Optimizations

In DRAM, the RB (i.e. sense amplifiers) and the bit-cells are tightly coupled due to charge sharing. Therefore, any read/write operation in DRAM requires the data to be fetched into the RB. A prominent characteristic of STT-RAM is the decoupled organization [13, 23, 29] of its sense amplifiers and the RB (Figure 5). This is a major advantage over DRAM for the following three reasons.

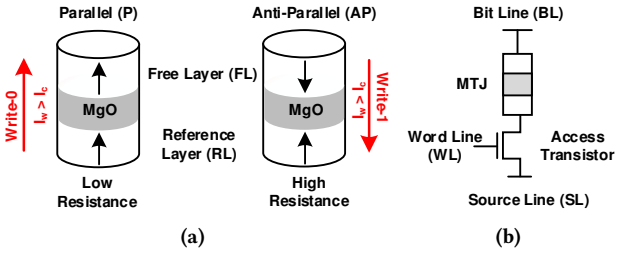


Figure 4: An STT-RAM bit-cell (a) and the Magnetic Tunnel Junction device (b) that stores a single bit of information.

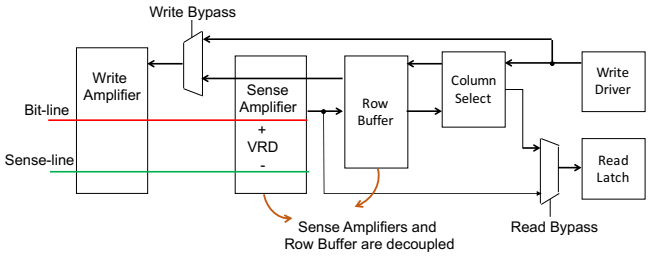


Figure 5: An STT-RAM peripheral circuit that supports both, memory access via a row buffer and direct access to the STT-RAM array via a bypass [13, 23].

(1) It is possible to bypass the RB in 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 [13, 23] to improve performance and energy efficiency by increasing the row buffer locality.
(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 requires the RB.
(3) The STT-RAM bank and the RB can operate independently on different rows which improves access parallelism.

Traditionally, on an RB miss, the newly requested row is always fetched in the row buffer. 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 writeback), the writeback access is particularly bad. An access after a writeback has a chance of less than 5% to hit the RB [13]. Therefore, recent work [13, 23] has proposed to bypass the RB for the writeback access and to perform the cache writeback operation directly on an STT-RAM column. This optimization improves both performance and energy efficiency by increasing RB hit rate and avoiding unnecessary RB evictions.

To further reduce the energy consumption, the partial write optimization proposed in [23] only writes dirty columns from the

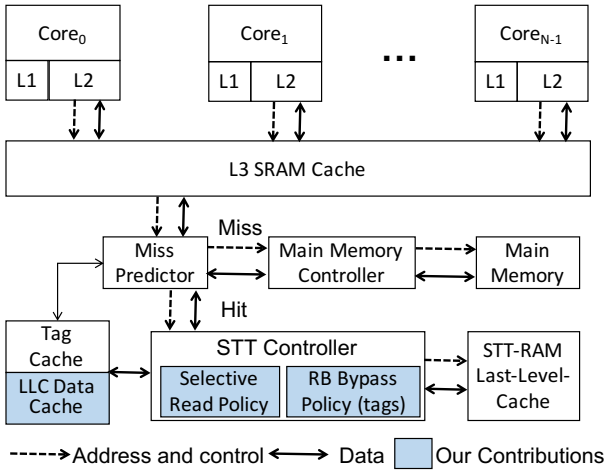


Figure 6: An STT-RAM based LLC architecture for an N-core system highlighting our contributions.

RB back to the STT-RAM array after an RB eviction. Both, the RB bypass and the partial write optimization are very easy to implement in STT-RAM due to its decoupled organization of the sense amplifiers and the RB as depicted in Figure 5.

3 PROPOSED STT-RAM LLC ARCHITECTURE

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

3.1 Overview

Figure 6 depicts an overview of the STT-RAM based LLC architecture and highlights our contributions. Similar to [13, 23], the proposed architecture uses RB bypass and partial write optimizations (recall Section 2.3) which improves performance and energy consumption compared to the traditional policy. Our LLC uses the same organization as the one proposed in [9, 10, 13] and shown in Figure 2.

A disadvantage of existing LLC RB policy [9, 10, 13, 27, 31] is that it always fetches an entire row into the RB while the majority of columns within the row are not accessed again. To reduce the energy consumption, we mark LLC lines as *lowly reused* and *highly reused* lines. Based on this distinction, we propose a *Selective Read Policy* that only fetches the useful STT-RAM columns into the RB. However, when we apply the *Selective Read Policy*, the existing tag read policy incurs high access latencies. To reduce this latency, we propose a novel *RB Tags Bypass Policy*. To further reduce the access latency and improve the LLC performance, we also propose the usage of a novel LLC Data Cache (LDC).

3.2 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 only fetch those cache lines that are likely to be reused, we avoid unnecessarily fetching 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 perform a simple experiment. For this, we use a state-of-the-art STT-RAM LLC with 2kB RB size, the LAMOST set mapping policy, and the STT-RAM optimizations discussed in Section 2.3. We run a series of SPEC2006 [2, 15] application mixes as listed in Table 2 and count the number of lines that are reused in the RB before eviction. Figure 1 displays the results of this experiment.

Figure 1 clearly shows that the majority of RB fetches are unnecessary, as none of the lines in the RB are accessed again in over 60% of the cases. The number of times that more then 4 lines in the RB are reused before eviction is less than 2%. This observation provides the motivation to fetch only those lines into the RB that are likely to be reused.

The RB has a low utilization due to the bank/row/set mapping (Figure 2b) of the LAMOST policy as shown in Figure 3. Since the super blocks that are mapped to the sets of the same LLC row are spacial distant, it is unlikely that subsequent accesses hit the RB. However, adjusting the mapping policy in favour of a higher RB hit rate would decrease the LLC hit rate. This trade-off in finding an ideal mapping policy is beyond the scope of this paper. We focus on improving the energy efficiency for the existing state-of-the-art LAMOST policy.

In order to only fetch lines into the RB that are likely to be reused, we classify the LLC lines as *highly-reused* and *lowly-reused* lines. We mark one line for each of the 4 sets within an LLC row as *highly-reused*. When selecting the *highly-reused* line of a set, we distinguish two cases. If a block of the currently requested super-block 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 super-block. If a set does not contain a block of the currently requested super-block, the least recently used (LRU) line of this set is the *highly-reused* line. This considers 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 Figure 7. We assume a super block S_b that contains four adjacent memory blocks named b_0, b_1, b_2, and b_3. The blocks are mapped to the sets Set_0, Set_1, Set_2, and Set_3, respectively. Further, we assume that 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 only fetch the 4 *highly-reused* lines.

In the example in Figure 7, 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 blocks b_0 and b_2 of the currently accessed super-block S_b. The other blocks of S_b (b_1 and b_3) are not currently present in the sets Set_1 and Set_3. Therefore, the LRU lines in these sets are marked as highly reused. In order to decide which line is *highly-reused*, the 4 tag columns need to be fetched first. Then, the 4 *highly-reused* lines are fetched into the RB. Our proposal requires a total of 8 column reads on an RB miss compared to 32 column reads in a traditional LLC architecture.

If a subsequent request misses the RB, the RB is evicted and the whole process is repeated. However, if a subsequent request hits the RB, we need to check if the requested line is present in the RB. Since we only fetch the *highly-reused* lines of each row, not all lines

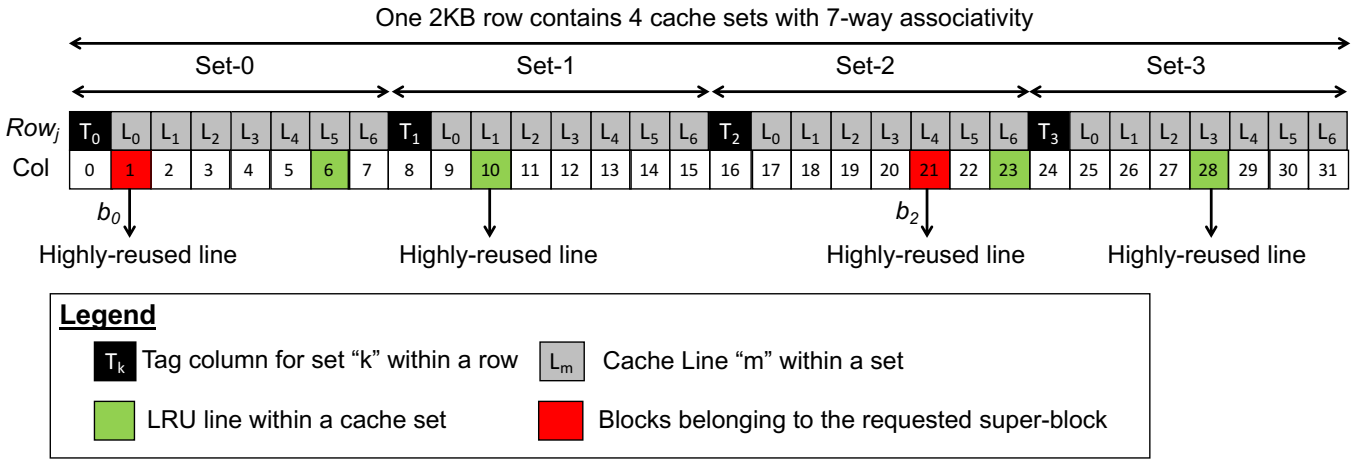


Figure 7: Example illustrating the *Selective Read Policy* using LAMOST organization and 2KB row size

are present in the RB. If the line is present in the RB, we can directly serve the request. Otherwise, we likely encountered the rare case in Figure 1 where more than 4 lines of a row are reused. Therefore, we now fetch the complete row from the STT-RAM array to the RB.

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 present bit for each line in the row. This requires 28 bits per STT-RAM bank. This is a total overhead of 224 bytes for our LLC organization with 64 banks.

3.3 Latency Breakdown

In this section, we analyze the latencies that occur in an STT-RAM based LLC architecture. We assume the LLC architecture depicted in Figure 6 along with a Tag-Cache. We further assume a request to block b_2 as in Figure 7. The latency breakdown for various scenarios in state-of-the-art LLC architectures is explained in the following:

RB miss and Tag-Cache miss in LAMOST. Figure 8a shows the latency breakdown of a RB and Tag-Cache 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 cache line, and two cycles to read the cache line (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 non-requested tag columns (i.e. T_0, T_1, and T_3) is 6 cycles. This extra latency overhead is repayed by future hits in the Tag-Cache leveraging the temporal locality of application. Therefore, the RB/Tag-Cache miss latency is 59 cycles.

RB miss and Tag-Cache hit. A Tag-Cache hit does not require an extra CAS command to access the tag column T_2, which is required to identify the location of the requested line. Therefore, read requests that hit in the Tag-Cache (Figure 8b) have a much lower latency (i.e. 40 cycles) compared to the Tag-Cache miss.

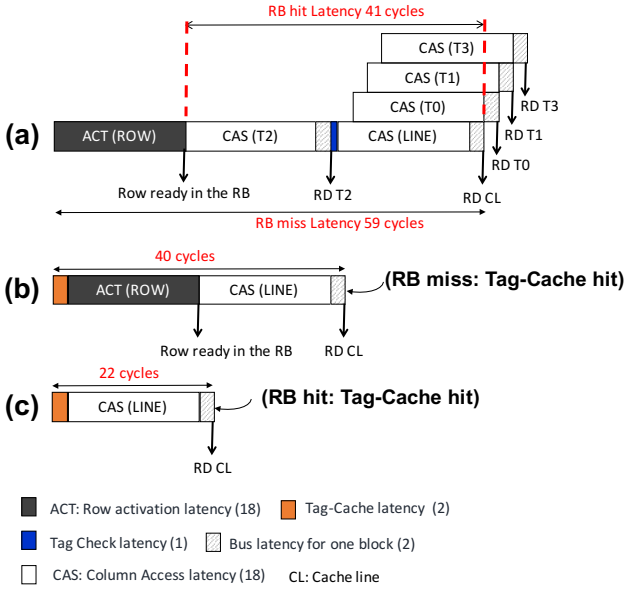


Figure 8: The latency incurred in *LAMOST* policy for various scenarios (a) RB miss and Tag-Cache miss (b) RB miss and Tag-Cache hit (c) RB hit and Tag-Cache hit

RB hit and Tag-Cache hit. Read requests that hit both in the RB and the Tag-Cache do not require an extra ACT command to fetch the requested row in the RB. Therefore, these requests are serviced much faster (i.e. 22 cycles) as illustrated in Figure 8c.

3.4 RB Tags Bypass Policy

The LLC tag read policy in [10, 11, 13, 16] always fetches the tags (and the lines) in the RB before inserting them in the Tag-Cache. However, this policy worsens the RB and Tag-Cache miss latency for the *Selective Read Policy* which is shown in Figure 9a. The *Selective Read Policy* requires two row activations. One for fetching the 4 tag columns and one subsequent activation for fetching the

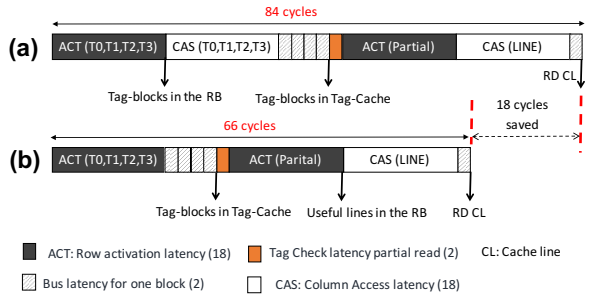


Figure 9: The RB miss and Tag-Cache miss latency with *Selective Read Policy* using (a) existing tag read policy (b) controller optimizations exploiting RB bypassing for the tags

Table 1: Latencies for read requests in various scenarios in our proposed LLC architecture excluding controller latency

RB Hit	Tag-Cache Hit	Cache Line Latency	Figure
No	No	66 cycles	9b
No	Yes	40 cycles	8b
Yes	No	41 cycles	8a
Yes	Yes	22 cycles	8c

highly-reused lines. Furthermore, additional 26 cycles (18 cycles for CAS and 8 cycles for the bus transfer) will be required to read all the tags from the RB into the Tag-Cache. Therefore, the RB miss and Tag-Cache miss latency is 84 cycles for the *Selective Read Policy* in combination with the existing tag reading policy.

The additional latency overhead for an RB and Tag-Cache miss using the *Selective Read Policy* is 25 cycles (84 - 59 = 25) when compared to the LAMOST policy. In order to reduce this latency overhead, we propose to bypass the RB for accessing the tags by exploiting the decoupled organization of the sense amplifiers and the RB in STT-RAM. Instead of fetching the tags into the RB, we store them directly in the Tag-Cache. Figure 9b shows the sequence of commands to read the tags and the cache line for this scenario. By avoiding the CAS command for fetching the tags from the RB, the latency is reduced by 18 cycles to a total of 66 cycles. Apart from reducing the access latency, the *RB Tags Bypass Policy* also reduces energy consumption, as the write energy for storing the tags in the RB and the read energy for reading the tags from the RB to the Tag-Cache are avoided.

To further improve performance, we update tags that are present directly in the Tag-Cahe and do not immediately write 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. In contrast to our approach, existing LLC architectures originally used for DRAM architectures [9, 10, 26, 27] always update the tags both in the Tag-Cache and the memory bank. In our implementation, the tags are only written back to the STT-RAM array after they are evicted from the Tag-Cache.

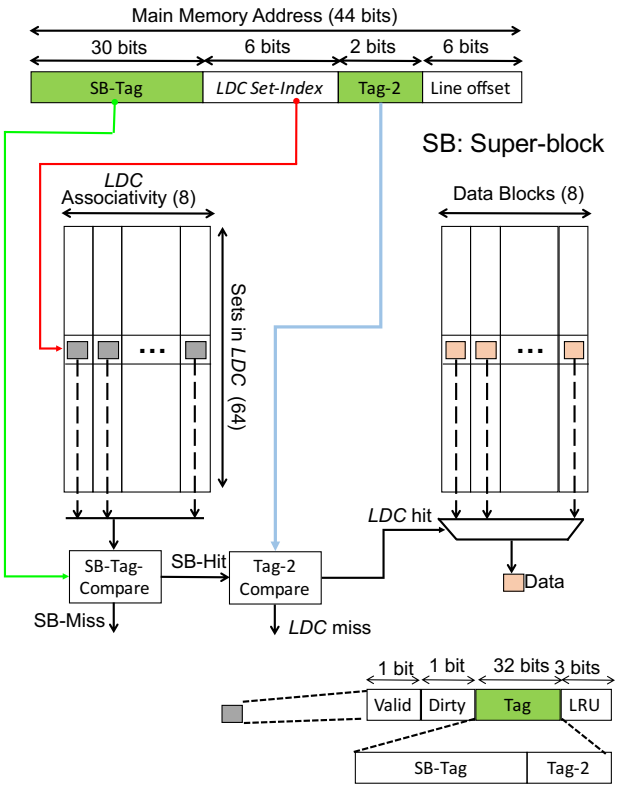


Figure 10: LLC Data Cache Organization (*LDC*)

3.5 LLC Data-Cache Organization

The access latency of reading a line from the STT-RAM array varies from 22 to 66 cycles depending on the scenario (see Table 1). To further reduce the latency, we extend our LLC architecture by a small on-chip SRAM structure called *LLC Data-Cache (LDC)* that holds the LLC cache lines that are likely to be accessed in the future. Figure 6 shows the addition of the *LDC* to the existing LLC architecture.

Figure 10 illustrates the organization of the LDC. It holds 64 sets with 8-way associativity and has a total size of 34.5 kB. LLC access that hit the *LDC* are serviced in only 2 cycles, due to the small structure. This accelerates *LDC* hits significantly compared to the 22 to 66 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 to the tag fields of the memory address to identify an *LDC* hit. The SB-Tag field of the memory address corresponds to the super-block and the Tag-2 field identifies the block within a super-block.

Figure 11 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

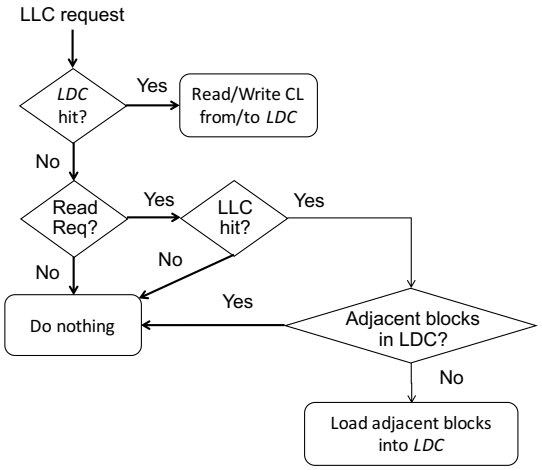


Figure 11: *LDC* lookup and insertion policy

hit, not the currently requested block is written to the *LDC*, but all adjacent blocks of the corresponding super-block are loaded to the *LDC*. The proposed *LDC* insertion policy exploits the temporal and spatial locality of applications as adjacent blocks of a super-block are likely be accessed in the near future. Note that all the blocks of the super-block (SB) are mapped to the same *LDC* set.

To elaborate our *LDC* insertion policy, we use the example from Figure 7. 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 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 blocks of the same super-block are very likely to be accessed in the near feature. Therefore, all adjacent blocks are 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.

Figure 12 shows the latency breakdown of an RB and Tag-Cache miss for the above example using the proposed *LDC* insertion policy in Figure 11. First, the LLC controller loads the tag columns from an STT-RAM array into the Tag-Cache and identifies the *highly-reused* lines (see Figure 7) of the row. Then, the *highly-reused* lines are loaded into the RB. Subsequently, read requests are sent to read the 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 2 cycles as illustrated in Figure 12. This latency overhead depends upon the number of adjacent blocks that are currently present in the LLC. It ranges from 0 to 6 cycles for 0 to 3 present blocks, respectively. The latency overhead for reading additional blocks to fill *LDC* is compensated by future hits in the *LDC*, exploiting the fact that these blocks will most likely be accessed later (confer to Figure 17 for evaluation).

3.6 Summary

In order to reduce the number of cache lines that are read unnecessarily to the RB, we introduced the *Selective Read Policy*. Using this policy, only one line of each of the 4 sets within a row is fetched into

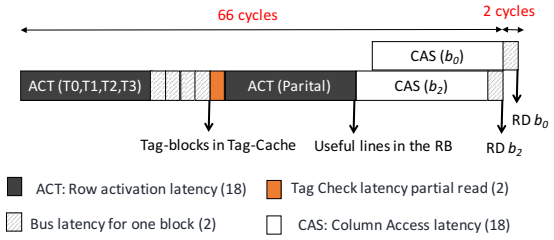


Figure 12: The RB/Tag-Cache miss latency with *LDC* using the proposed *LDC* insertion policy in Figure 11 for the illustrating example in Figure 7

the row buffer. These 4 lines are likely to be reused on a subsequent access.

For each line in a bank we store a presence bit to keep track of the lines currently residing in that bank. If an access hits the RB but the corresponding line is not present in the RB, the whole row is fetched to the RB. (i.e. less than 2% of the time for all application mixes).

In order to identify the *highly-reused* line of a set, first the tag columns of the row need to be analyzed. To reduce the latency for tag column reads and to further reduce the energy consumption, we do not load the tag columns into the RB but load them directly to the Tag-Cache. This is referred to as *RB Tags Bypass Policy*. In addition, we update present tags directly in the Tag-Cache and write them back to the STT-RAM array only after eviction from the Tag-Cache.

As shown in Table 1, the service times for LLC requests lies in a range from 22 to 66 cycles. To further exploit the temporal and spacial locality of typical applications, we introduced an additional *LLC Data Cache (LDC)* that holds lines that are likely to be used in future requests. A hit in the LDC, reduces the service time for the request to only 2 cycles. Similar to the work in [13], our STT-RAM architecture bypasses the RB and Tag-Cache for writebacks. Further, we do not insert lines into the *LDC* for writes and writebacks.

4 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.

4.1 Experimental Setup

We evaluate our STT-RAM LLC architecture using the Zesto X86 simulator [28]. Zesto provides detailed cycle accurate models of the core mirco-architecture and of the cache hierarchy. For our benchmarks, we model an 8-core system where each core runs a single application from the SPEC2006 benchmark suite [2, 15]. We simulate a total of 6 application mixes as shown in Table 2.

The parameters of the system configuration used for simulation are listed in Table 3. We extensively modified the DDR memory model of Zesto to reflect the distinct characteristics of STT-RAM similar to the work in [13, 23]. Most importantly, we model the non-volatility and high write latency of STT-RAM. The modified

Table 2: 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 3: Configuration details as used in the experiments

Core	3.2 GHz, out-of-order, 4-issue
Private L1 Private L2 Shared L3	32 kB, 8-way associativity, 2 cycles latency 512 kB, 8-way associativity, 5 cycles latency 8 MB, 8-way associativity, 20 cycles latency
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 (cycles)
t_{WR}	18 and 38 cycles for DRAM and STT-RAM
Tag-Cache	27 kB, 2 cycle latency [10, 11, 16]
Miss Predictor	Map-I [31], 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 (cycles)

STT-RAM LLC model considers bus contention, queuing delays, as well as bank and row buffer conflicts. We extracted the energy values of various micro-architectural structures using NVSIM [5] and Cacti [30, 37] and configured Zesto accordingly. Table 4 shows the energy values for various operations. These values, include the energy consumed by, e.g., row and column decoders, sense amplifiers, multiplexers, write drivers, and read latches.

For evaluation, we compare the following LLC configurations:

DRAM-Base DRAM LLC without any optimizations [9, 10].

STT-Base STT-RAM LLC without any optimizations.

STT-WP STT-RAM LLC including the state-of-the-art write-back RB bypass [13] and partial write optimizations [23] described in Section 2.3.

STT-WP-SR-naive State-of-the-art STT-RAM LLC extended by our *Selective Read Policy* as described in Section 3.2 using existing tag read policy in [10, 11].

STT-WP-SR-TagOpt State-of-the-art STT-RAM LLC extended by our *Selective Read Policy* using the *RB Tags Bypass Policy* from Section 3.4.

STT-WP-SR-TagOpt-LDC STT-WP-SR-TagOpt extended by the *LLC Data Cache (LDC)* from Section 3.5.

Table 4: Dynamic energy consumption of various components detailed in Table 3

Operation	Energy
DRAM Array Read/Write	14.13 [nJ/Column]
DRAM Precharge	4.63 [nJ/Column]
DRAM RB Access	11.88 [nJ/Column]
STT-RAM Array Read	10.87 [nJ/Column]
STT-RAM Array Write	28.49 [nJ/Column]
STT-RAM RB Access	10.07 [nJ/Column]
Tag-Cache Access (27.2 KB)	0.97 [nJ/Access]
LLC Data Cache (LDC) (34.5 KB)	1.12 [nJ/Access]

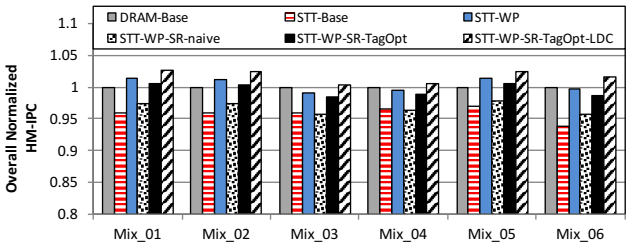


Figure 13: Measured harmonic mean instruction per cycle (HM-IPC) for the 6 application mixes and different configurations

For all configurations, we make the following assumptions:

(1) We use the LAMOST LLC set mapping policy [9, 10] (see Figure 2)
(2) We employ a MAP-I predictor for predicting LLC cache misses as in [31] and Tag-Cache as in [10, 11, 16].
(3) We assume an LLC with 4 channels, 256 bits bus width per channel, 2 cycles bus latency, and 2kB RB size.
(4) We assume that writes to the STT-RAM array incur 20 cycles extra latency compared to a DRAM array. We assume t_{WR} (write recovery time) to be 18 cycles for DRAM and 38 cycles for STT-RAM.
(5) We employ the Simpoint tool [14] to choose the region of interest (ROI) for each benchmark in the application mix.
(6) We use FR-FCFS (First Ready First Come First Serve) access scheduling [32] for the LLC.

4.2 Performance and Energy Measurements

Figure 13 shows the performance results for all evaluated configurations normalized to DRAM-Base configuration. As depicted, our STT-WP-SR-TagOpt-LDC configuration improves the average performance by 1.4%, 5.6%, and 1.3% compared to DRAM-Base, STT-Base, and STT-WP configurations, respectively.

The energy benefits of the STT-WP-SR-TagOpt-LDC configuration can be observed in Figure 14. As shown, the average energy consumption is reduced by 90.6% compared to DRAM-base and 91.2% compared to STT-Base. While the writeback RB bypass policy proposed in [13] already saves a large amount of energy, our

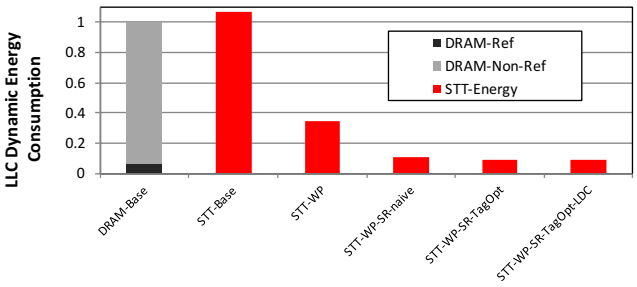


Figure 14: Average LLC dynamic energy consumption for various configurations normalized to DRAM-Base configuration [9, 10]

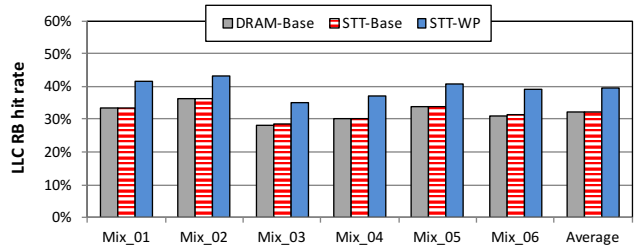


Figure 15: LLC RB hit rate for various configurations

Selective Read Policy further reduces this energy consumption by 72.2% by eliminating unnecessary STT-RAM reads.

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

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 to DRAM. This is mostly due to the high latency and energy consumption of write operations in STT-RAM. However, due to the decoupled organization of 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.

Comparing STT-Base and STT-WP. Applying the writeback RB bypass policy [13], significantly improves the performance and energy consumption of an STT-RAM LLC. In the 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 Figure 15. 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 [23]. As a consequence of both optimizations, the LLC dynamic energy consumption is reduced by 69.2% compared to the STT-Base configuration.

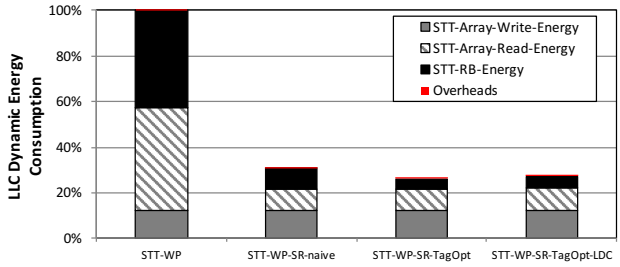


Figure 16: Average LLC dynamic energy consumption of various configurations. Values are normalized to the STT-WP configuration for reference. Overheads include the energy consumed by the Tag-Cache and the *LDC*

Impact of the Selective Read Policy. Although the STT-WP configuration is more energy efficient than the 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 3.2) further reduces the energy consumption by 72.2% compared to the STT-WP configuration. Figure 16 illustrates the energy improvement of configurations that use the *Selective Read Policy*. However, the *Selective Read Policy* degrade the performance by an average of 3.6% compared to the STT-WP configuration (see Figure 13) due to an increased RB and Tag-Cache miss latency (see Figure 9a).

Impact of the RB Tags Bypass Policy. Since the *Selective Read Policy* suffers from an increased RB and Tag-Cache miss latency, it has a limited performance. We proposed the *RB Tags Bypass Policy* (Section 3.4), which bypasses the RB for tag access, to overcome this limitation. As shown in Figure 9, this optimization reduces the latency from 84 to 66 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-TagOpt configuration improves the average performance by 2.9% and reduces the LLC dynamic energy consumption 14.7% compared to the STT-WP-SR-naive configuration.

Impact of the LDC. The STT-WP-SR-TagOpt configuration improves performance compared to the STT-WP-SR-naive configuration. However, it still degrades the performance by 0.8% on average compared to the state-of-the-art STT-WP configuration. To further improve the performance, we proposed a small structure called *LDC Data Cache (LDC)* (see Section 3.5). The inclusion of *LDC* improves the performance by 1.3% and 2% compared to the STT-WP and STT-WR-SR-TagOpt configurations. The proposed *LDC* accelerates access latency to cache lines as an STT-RAM read can be avoided when the line is present in the *LDC*. As illustrated in Figure 17, the LLC hit for a read request is 47.5% on average for all application mixes. On the downside, the STT-WP-SR-TagOpt-LDC configuration increases LLC dynamic energy consumption by 4.3% compared to the STT-WP-SR-TagOpt configuration due to reading extra columns from the STT-RAM array.

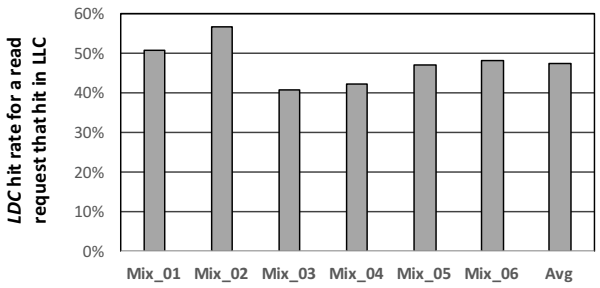


Figure 17: *LDC* hit rate for a read request that hits in LLC

5 RELATED WORK

Existing LLC architectures (mostly DRAM-based) can be catergorized into block-based and page-based designs. Block-based LLCs [9–11, 16, 25, 27, 31, 33, 34] use a small line size (i.e. 64 byte) while page-based LLCs [17–20, 40] use a large line size (i.e. 1KB/2KB). 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 multi-core system running diverse applications. Another drawback of page-based LLC is that not all 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, IBM POWER7 processor [3, 39] employs a 32 MB DRAM LLC. The DRAM based LLC architectures [9–11, 16–20, 25, 27, 31, 33, 34, 40] 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 DRAM, recently STT-RAM LLC architecture [13] have been proposed as a promising alternative to DRAM LLC.

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

Many architectural techniques have been presented for energy and performance trade-offs in STT-RAM caches [21, 35, 36]. They relax the non-volatility of 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 larger STT-RAM due to high energy requirements for refreshing MTJ cells periodically. In contrast, our proposal exploits the non-volatility characteristics of 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 [4, 42]. However, these circuit level techniques are orthogonal to our work and can be combined with our proposal in order to further improve performance and energy efficiency of STT-RAM LLC.

6 CONCLUSIONS

This paper presents novel policies to improve the performance and energy efficiency in STT-RAM LLC architectures. We demonstrate that existing LLC architectures unnecessarily fetch large amounts of data into the row buffer while the majority of the data is not reused. Our *Selective Read Policy* exploits this fact to reduces the energy consumption by decreasing the number of STT-RAM reads. While this introduces a latency penalty, we decrease the access latency by a new *Row Buffer Tags Bypass Policy* optimization. Finally, the *LLC Data Cache* (*LDC*) organization is proposed to improve the performance by further reducing the access latency. Our results on SPEC 2006 benchmarks show that our synergestic policies are effective in improving the average performance (1.3% and 1.4%) and energy consumption (90.6% and 72.6%) compared to the state state-of-the-art proposal for DRAM and STT-RAM LLC. With these optimizations STT-RAM becomes an effective Last-Level-Cache alternative to DRAM.

ACKNOLEDGMENTS

This work was partly supported by the German Research Foundation (DFG) within the Cluster of Excellence 'Center for Advancing Electronics Dresden' (cfaed).

REFERENCES

[1] 2013. Hybrid Memory Cube Consortium: Hybrid Memory Cube Specification. http://www.jedec.org/standards-documents/docs/jesd235. (2013).
[2] 2017. Standard Performance Evaluation Corporation. http://www.spec.org. (2017). [Online; accessed 10-March-2017].
[3] R. X. Arroyo, R. J. Harrington, S. P. Hartman, and T. Nguyen. 2011. IBM POWER7 Systems. *IBM Journal of Research and Development* 55, 3 (2011), 2:1 – 2:13.
[4] R. Bishnoi, F. Oboril, M. Ebrahimi, and M.B. Tahoori. 2014. Avoiding Unnecessary Write Operations in STT-MRAM for Low Power Implementation. In *Proceedings of the 15th International Symposium on Quality Electronic Design (ISQED'14)*. 548–553.
[5] X. Dong, C. Xu, Y. Xie, and N.P. Jouppi. 2012. NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31, 7 (July 2012), 994–1007.
[6] Darryl Gove. 2007. CPU2006 Working Set Size. *SIGARCH Computer Architecture News* 35, 1 (March 2007), 90–96.
[7] Fazal Hameed, L. Bauer, and J. Henkel. 2013. 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)*. 77–82.
[8] Fazal Hameed, L. Bauer, and J. Henkel. 2013. Reducing Inter-Core Cache Contention with an Adaptive Bank Mapping Policy in DRAM Cache. In *IEEE International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'13)*.
[9] Fazal Hameed, L. Bauer, and J. Henkel. 2013. 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)*.
[10] Fazal Hameed, L. Bauer, and J. Henkel. 2014. Reducing Latency in an SRAM/DRAM Cache Hierarchy via a Novel Tag-Cache Architecture. In *Proceedings of the 51st Design Automation Conference (DAC'14)*.
[11] Fazal Hameed, L. Bauer, and J. Henkel. 2016. Architecting On-Chip DRAM Cache for Simultaneous Miss Rate and Latency Reduction. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35, 4 (April 2016), 651–664.
[12] Fazal Hameed and Jeronimo Castrillon. 2017. Rethinking On-chip DRAM Cache for Simultaneous Performance and Energy Optimization (to appear). In *Proceedings of the 19th conference on Design, Automation and Test in Europe (DATE)*.

[13] Fazal Hameed and M. B. Tahoori. 2016. Architecting STT Last-Level-Cache for Performance and Energy Improvement. In *2016 17th International Symposium on Quality Electronic Design (ISQED)*. 319–324.

[14] G. Hamerly, E. Perelman, J. Lau, and B. Calder. 2005. SimPoint 3.0: Faster and More Flexible Program Analysis. *Journal of Instruction Level Parallelism* 7 (2005).

[15] John L. Henning. 2006. SPEC CPU2006 Benchmark Descriptions. *SIGARCH Computer Architecture News* 34, 4 (September 2006), 1–17.

[16] C-C. Huang and V. Nagarajan. 2014. 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)*. 51–60.

[17] D. Jevdjic, G.H. Loh, C. Kaynak, and B. Falsafi. 2014. Unison Cache: A Scalable and Effective Die-Stacked DRAM Cache. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 25–37.

[18] D. Jevdjic, S. Volos, and B. Falsafi. 2013. 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)*. 404–415.

[19] X. Jiang, N. Madan, L. Zhao, M. Upton, R. Iyer, S. Makineni, D. Newell, Y. Solihin, and R. Balasubramonian. 2010. CHOP: Adaptive Filter-Based DRAM Caching for CMP Server Platforms. In *Proceedings of the 16th IEEE Symposium on High-Performance Computer Architecture (HPCA)*. 1–12.

[20] X. Jiang, N. Madan, L. Zhao, M. Upton, R. Iyer, S. Makineni, D. Newell, Y. Solihin, and R. Balasubramonian. 2011. CHOP: Integrating DRAM Caches For CMP Server Platforms. *IEEE Micro Magazine (Top Picks), IEEE Computer Society* (2011), 99–108.

[21] A. Jog, A.K. Mishra, Cong Xu, Y. Xie, V. Narayanan, R. Iyer, and C.R. Das. 2012. Cache Revive: Architecting Volatile STT-RAM Caches for Enhanced Performance in CMPs. In *Proceedings of the 49th IEEE/ACM Design Automation Conference (DAC '12)*. 243–252.

[22] U. Kang, H.-J. Chung, S. Heo, S.-H. Ahn, H. Lee, S.-H. Cha, J.D. Ahn, J.H. Kim, J.-W. Lee, H.-S. Joo, W.-S. Kim, H.-K. Kim, E.-M. Lee, S.-R. Kim, K.-H. Ma, D.-H. Jang, N.-S. Kim, M.-S. Cho, S.-J. Oh, J.-B. Lee, T.-K. Jung, J.-H. Yoo, and C. Kim. 2010. 8 Gb 3-D DDR3 DRAM using Through-Silicon-Via Technology. In *IEEE Journal of Solid State Circuits*, Vol. 45. 111–119.

[23] E. Kultursay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu. 2013. Evaluating STT-RAM as an Energy-Efficient Main Memory Alternative. In *International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 256–267.

[24] D. U. Lee, K. W. Kim, K. W. Kim, H. Kim, J. Y. Kim, Y. J. Park, J. H. Kim, D. S. Kim, H. B. Park, J. W. Shin, J. H. Cho, K. H. Kwon, M. J. Kim, J. Lee, K. W. Park, B. Chung, and S. Hong. 2014. 25.2 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 *Proceedings of the International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. 432–433.

[25] G.H. Loh. 2009. Extending the Effectiveness of 3D Stacked DRAM Caches with an Adaptive Multi-Queue Policy. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 174–183.

[26] G.H. Loh and M.D. Hill. 2011. Efficiently Enabling Conventional Block Sizes for Very Large Die-stacked DRAM Caches. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 454–464.

[27] G.H. Loh and M.D. Hill. 2012. Supporting Very Large DRAM Caches with Compound Access Scheduling and MissMaps. *IEEE Micro Magazine, Special Issue on Top Picks in Computer Architecture Conferences* 32, 3 (2012), 70–78.

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

[29] J. Meza, L. Jing, and O. Mutlu. 2012. A Case for Small Row Buffers in Non-volatile Main Memories. In *Proceedings of the 30th International Symposium on Computer Design(ICCD)*. 484–485.

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

[31] M.K. Qureshi and G.H. Loh. 2012. Fundamental Latency Trade-offs in Architecting DRAM Caches. In *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 235–246.

[32] S. Rixner, W. Dally, U. Kapasi, P. Mattson, and J. Owens. 2000. Memory Access Scheduling. In *Proceedings of the 32nd International Symposium on Computer Architecture (ISCA)*. 128–138.

[33] J. Sim, G.H. Loh, H. Kim, M. O??Connor, and M. Thottethodi. 2012. A Mostly-Clean DRAM Cache for Effective Hit Speculation and Self-Balancing Dispatch. In *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 247–257.

[34] J. Sim, G.H. Loh, V. Sridharan, and M. O'Connor. 2013. Resilient die-stacked DRAM caches. In *Proceedings of the 40th International Symposium on Computer Architecture (ISCA)*.

[35] C.W. Smullen, V. Mohan, A. Nigam, S. Gurumurthi, and M.R. Stan. 2011. Relaxing Non-volatility for Fast and Energy-efficient STT-RAM Caches. In *Proceedings of the 17th IEEE Symposium on High-Performance Computer Architecture (HPCA)*. 50–61.

[36] Z. Sun, X. Bi, H.H. Li, W-Fai Wong, Z-Liang Ong, X. Zhu, and W. Wu. 2011. Multi Retention Level STT-RAM Cache Designs with a Dynamic Refresh Scheme. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '11)*. 329–338.

[37] S. Thoziyoor, J.H. Muralimanohart, R.and Ahn, and N. Jouppi. 2008. CACTI 5.1 HPL 2008/20, HP Labs. (April 2008).

[38] Christian Weis, Matthias Jung, and Nobert Wehn. 2016. 3D Memories. *Book chapter in the Handbook of 3D Integration* 4 (2016).

[39] D. Wendel, R. Kalla, R. Cargoni, J. Clables, J. Friedrich, R. Frech, J. Kahle, B. Sinharoy, W. Starke, S. Taylor, S. Weitzel, S.G. Chu, S. Islam, and V. Zyuban. 2010. The Implementation of POWER7TM: A Highly Parallel and Scalable Multi-core High-end Server Processor. In *Proceedings of the International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. 102–103.

[40] D.H. Woo, N.H. Seong, D.L. Lewis, and H-H.S. Lee. 2010. An Optimized 3D-stacked Memory Architecture by Exploiting Excessive, High-density TSV Bandwidth. In *Proceedings of the 16th IEEE Symposium on High-Performance Computer Architecture (HPCA)*. 1–12.

[41] W.A. Wulf and S.A. McKee. 1995. Hitting the Memory Wall: Implications of the Obvious. *Computer Architecture News* 23, 1 (March 1995), 20–24.

[42] P. Zhou, B. Zhao, J. Yang, and Y. Zhang. 2009. Energy Reduction for STT-RAM Using Early Write Termination. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '09)*. 264–268.