IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 15, 2022, accepted March 5, 2022, date of publication March 10, 2022, date of current version March 21, 2022.

*Digital Object Identifier 10.1109/ACCESS.2022.3158493*

# Proactively Invalidating Dead Blocks to Enable Fast Writes in STT-MRAM Caches

**YONGJUN KIM[1], YUZE CHEN[2], YONGHO LEE[1], LIMEI PENG[2], AND SEOKIN HONG[1]**
[1]Department of Electrical and Computer Engineering, Sungkyunkwan University, Suwon 16419, Republic of Korea
[2]School of Computer Science and Engineering, Kyungpook National University, Daegu 41566, Republic of Korea

Corresponding author: Seokin Hong (seokin@skku.edu)

This work was supported in part by the Institute of Information Communications Technology Planning Evaluation (IITP) grant funded by the Korea Government (MSIT), Intelligent in-memory error-correction device for high-reliability memory, 50%, under Grant 2021-0-00863; and in part by the BK21 FOUR Project (50%).

**ABSTRACT** Spin-Transfer Torque Magnetic Random Access Memory (STT-MRAM) is a promising emerging memory technology for on-chip caches. It has a low read access time and low leakage power. Unfortunately, however, STT-MRAM suffers from its long write latency and high write energy consumption. This paper proposes a cache management technique called Proactive Invalidation (PROI) that proactively invalidates dead blocks in advance to enable fast writes in the STT-MRAM caches. Experimental evaluation shows that the proposed technique improves performance by 14% on average compared to the baseline STT-MRAM cache. This paper also proposes two optimization techniques called Proactive Invalidation-aware Data Encoding (PIDE) and Narrowness-aware Partial Write (NPW) to minimize the energy overheads of Proactive Invalidation. Experimental results demonstrate that the total energy consumption of the STT-MRAM cache with PROI is only 1.8% higher than the baseline when PROI is applied with PIDE and NPW.

**INDEX TERMS** On-chip cache, non-volatile memory, STT-MRAM, dead block, narrow-width value.

## I. INTRODUCTION

Modern processors integrate multiple cores to meet the high-performance and low-power computing requirements. As the number of cores in a processor increases and the working set size of the applications increases, larger on-chip caches are required for the multi-core processor. However, the current on-chip caches have a limit on scalability because of the high leakage power consumption and large cell size of the SRAM, which is a conventional memory technology. Recently, non-volatile memories have been considered as an alternative to traditional memory technologies such as SRAM and DRAM. The Spin-transfer torque random access memory (STT-MRAM) is one of the emerging non-volatile memory technologies that can be used for on-chip caches since its access time is comparable to SRAM, and it has low leakage power [1], [2]. Furthermore, STT-MRAM is considerably smaller than SRAM, enabling it to contain more data within the same chip area. Therefore, STT-MRAM could be a promising memory technology for large last-level caches (LLC) with these attractive characteristics.

The associate editor coordinating the review of this manuscript and approving it for publication was Thomas Canhao Xu.

Unfortunately, however, write latency and write power consumption of the STT-MRAM are much higher than those of the SRAM [3]. To tackle these limitations of the STT-MRAM, many prior works have proposed circuit-level or architecture-level techniques to reduce the write operations on the STT-MRAM caches or reduce the write energy of the STT-MRAM cell [4]–[14]. In many architecture-level techniques, the LLC comprises SRAM banks and STT-MRAM banks and allocates the write-intensive cache blocks (i.e., frequently updated cache blocks) to the SRAM banks [6]–[11]. This approach's effectiveness can vary depending on the fraction of the write-intensive cache blocks and the size of the SRAM banks. The circuit-level techniques reduce the write energy of the STT-MRAM by terminating the write operations as soon as data is written to the STT-MRAM [4], [5]. Even if this approach reduces the write energy consumption of the STT-MRAM caches, it can not reduce the write latency.

In this paper, we propose a novel last-level cache (LLC) management technique called Proactive Invalidation (PROI) that exploits an asymmetric characteristic in the write operation of the STT-MRAM to address its long write latency. The write latency of the STT-MRAM is different depending

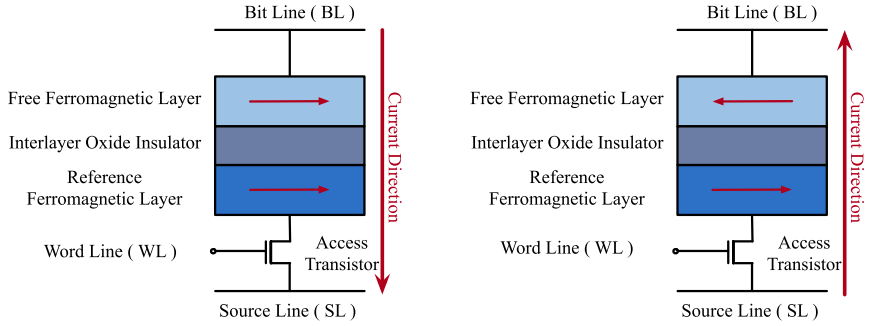



**FIGURE 1.** STT-MRAM structure: (1a) MTJ switches to parallel state when applying a write current from bit line to source line. (1b) MTJ switches to anti-parallel state when applying a write current from source line to bit line.

on the type of resistance state transition. The write latency is high when an STT-MRAM's resistance state is changed to the anti-parallel state from the parallel state. On the other hand, when the resistance state is changed to the parallel state from the anti-parallel state, the write latency is low [4], [15]. To exploit this characteristic, PROI proactively invalidates cache blocks if predicted as dead blocks. When invalidating the dead block, PROI switches all STT-MRAM cells of the corresponding cache entry to the anti-parallel state. If a cache entry is proactively invalidated, there will be no slow transitions (i.e., parallel states to anti-parallel states), resulting in fast writes on the cache entry.

Even if PROI can enhance the performance, it can increase write energy consumption due to the additional writes involved in the invalidation operations. To reduce the impact of PROI on the write energy consumption, we propose two optimization techniques: Proactive Invalidation-aware Data Encoding (PIDE) and Narrowness-aware Partial Write (NPW). In many prior works, the parallel state represents binary '0,' and the anti-parallel state represents binary '1'. However, we found that this data encoding scheme is inefficient in terms of energy consumption when applying the PROI. PIDE encodes data in such a way that minimizes the frequency of the state transitions to the anti-parallel state by considering the distribution of binary '0' and '1' in data stored in the LLC. The second optimization technique, called NPW, avoids the redundant write operations to reduce energy consumption further. It is well known that a wide range of applications frequently uses narrow-width values whose upper bits are all zeros [16]–[19]. By exploiting narrow-width values, NPW avoids unnecessary write operations on some STT-MRAM cells if the data stored in the cache entry is narrow-width.

Experimental evaluation using a system-level and processor simulator with SPEC CPU2006 [20] and PARSEC [21] benchmark suites show that PROI achieves a performance improvement of 14% on average with a small energy overhead (1.8%) compared to a baseline STT-MRAM cache.

In this paper, we make **three main contributions**

1) We propose a new cache management technique for the STT-MRAM caches by exploiting the inherent physical properties of the STT-MRAM cell and characteristics of the LLC data access pattern.
2) We propose two optimizations to reduce the impact of the proposed technique on energy consumption. In the first optimization, data is encoded before written to the cache in such a way that minimizes the frequency of the transitions to an anti-parallel state. The second optimization partially updates cache contents by exploiting the narrowness of the data written to the caches.
3) We evaluate the performance and energy efficiency of the proposed technique with a system-level and processor simulator with the multi-programmed and multi-threaded workloads.

## II. BACKGROUND AND MOTIVATION

### A. STT-MRAM

Spin transfer torque magnetic random access memory (STT-MRAM) is a promising emerging non-volatile memory technology where data is stored in a ferromagnetic layer of MTJ (Magnetic Tunnel Junction). STT-MRAM has advantages such as higher density, non-volatility, high soft error endurance, and low power consumption [22]. However, although STT-MRAM has many attractive features, it suffers from high write energy consumption and high write latency.

In STT-MRAM, data stored in a cell is represented as two different resistance states of the MTJ. As shown in Figure 1, the MTJ consists of a thin interlayer oxide insulator (MgO) and two ferromagnetic layers: free and reference layers. The magnetization direction of the free ferromagnetic layer can be changed. On the contrary, the magnetization direction of the reference layer is not changed. By applying a write current between the source line (SL) and the bit line (BL), we can change the magnetization direction of the free layer.

The resistance of an MTJ is determined by the relative magnetization directions of the two ferromagnetic layers. If the magnetic directions of the two layers are different (i.e., anti-parallel state), the resistance of the MTJ is high. In contrast, if the magnetic directions of the two layers are the same (i.e., parallel state), the resistance of the MTJ is low.

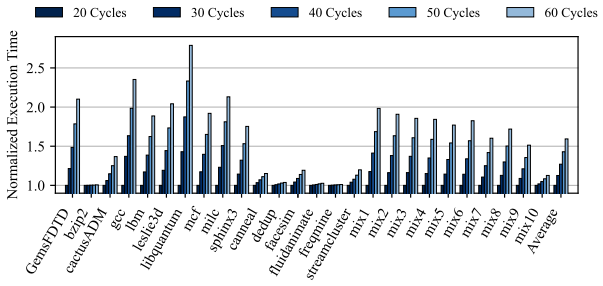



**FIGURE 2.** Performance impact of the LLC's write latency.

Thus, we can use one of two states of the MTJ to represent binary '0' or '1'.

### B. A CHALLENGE: WRITES ON STT-MRAM ARE EXPENSIVE

As explained in section II-A, the STT-MRAM suffers from long write latency and high write energy consumption even if its read latency and read energy consumption are similar to those of the SRAM [23]. The long write latency has a significant negative impact on cache performance because the long write operations prevent subsequent cache accesses, reducing the cache bandwidth and potentially offsetting the capacity benefit of the STT-MRAM. Furthermore, the long write latency also leads to significant energy consumption as a high switching current flows through the STT-MRAM cell until it is switched to the target resistance state [15].

Figure 2 shows the impact of the LLC's write latency on the system performance. As shown in the figure, the execution time significantly increases for most memory-intensive workloads as the LLC's write latency increases. On average, when the LLC's write latency increases from 20 cycles to 60 cycles, the execution time increases by 59%. This result demonstrates that the impact of LLC's write latency on performance is undoubtedly significant and critical for system performance.

### C. OPPORTUNITIES FOR REDUCING STT-MRAM COSTS

To mitigate the long write latency and high energy consumption of the STT-MRAM when designing STT-MRAM-based LLCs, we exploit circuit-, architecture-, and software-level characteristics.

#### 1) CIRCUIT-LEVEL CHARACTERISTIC: ASYMMETRICAL WRITE LATENCY

STT-MRAM has an asymmetry in the write latency when switching it to either the parallel or anti-parallel state [4], [15], [24]. Switching the STT-MRAM to the parallel state is faster than switching it to the anti-parallel state. This asymmetry in the STT-MRAM's write latency is mainly due to asymmetric characteristics of the MTJ and the access transistor. The MTJ has an inherent torque asymmetry since it has two different mechanisms to switch the magnetic orientation of the free ferromagnetic layer. When switching the MTJ to the parallel state, the same spin direction electrons as the magnetic orientation of the reference layer are applied.

**TABLE 1.** MTJ switching latency [4].

| | With access transistor | Without access transistor |
|---|---|---|
| Switch to parallel state | 3.9 ns | 3.9 ns |
| Switch to anti-parallel state | 9.7 ns | 5.7 ns |

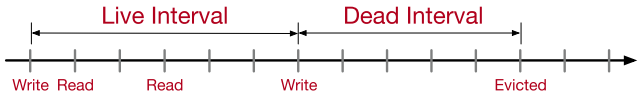



**FIGURE 3.** Life cycle of cache block.

In contrast, switching the MTJ to the anti-parallel state is performed by applying the opposite spin direction electrons. When the spin direction of the electrons is opposite to the magnetic orientation of the reference layer, they are reflected at the boundary of the interlayer oxide insulator and the reference ferromagnetic layer [25]. Thus, switching the MTJ to the anti-parallel state takes more time and consumes more energy than switching it to the parallel state. The voltage degradation in the access transistor also contributes to the asymmetric characteristic of the STT-MRAM in the write latency. With the voltage degradation in the access transistor, write current is reduced while switching the MTJ to anti-parallel state [4].

In general, as shown in Table 1, without the access transistor, the write latency for switching the MTJ to the anti-parallel state is 1.46 times greater than the writing latency for switching the MTJ to the parallel state. With the access transistor, the write latency for switching the MTJ to the anti-parallel state is 2.48 times greater than the latency for switching the MTJ to the parallel state.

The asymmetry in the write latency gives us an opportunity to reduce the write latency of the STT-MRAM caches. As shown in Table 1, switching to the anti-parallel state are much slower than switching to the parallel state. Therefore, if we eliminate the state transition to the anti-parallel state (i.e., slow writes) as much as possible from the critical path of the write operations in the STT-MRAM caches, we can reduce their overall write latency.

#### 2) ARCHITECTURE-LEVEL CHARACTERISTIC: DEAD BLOCK

Dead block has been studied well and exploited in many prior works to improve the efficiency of the on-chip caches [26]–[29]. The dead block is a cache block that is not going to be referenced shortly before being evicted from the on-chip caches. After the cache blocks are stored in the caches, they are generally accessed at a regular interval, and then they will remain in the cache without any re-references as shown in Figure 3. In the LLCs, only a small fraction of the cache blocks hold data that will be referenced before eviction [28] and most of them stay in the cache without any re-references. The dead blocks lead to poor cache efficiency because they waste the capacity and energy of the LLCs. They occupy many cache entries of the LLC and move from the MRU

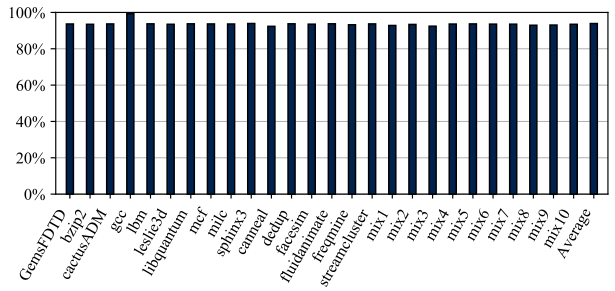



**FIGURE 4.** Percentage of dead LRU blocks.

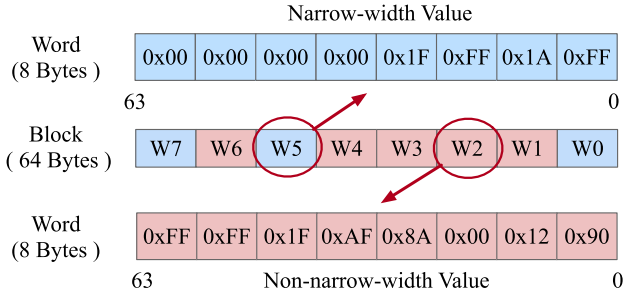



**FIGURE 5.** Cache block with narrow-width value.

(most recently used) to the LRU (least recently used) position before it is evicted from the cache [26].

In general, the dead blocks are predicted by learning the past access pattern of the cache blocks. Various dead block prediction algorithms have been proposed [27], [29], [30]. Trace-Based Predictor [27] collects traces of the instruction addresses that access a particular block. If the trace results in the last access to one block, the same trace will result in the last access to other blocks. Some cache blocks are predicted as dead by indexing the sum of these instruction addresses. Time-based dead block predictor [30] predicts a block as dead if it is not accessed more than twice the live time by collecting each block's number of live cycles. Counting-based predictor [29] predicts the dead block if a block has been accessed more times than the previous generation or has been accessed the same number of times in the last two generations. To this end, it employs a predictor table composed of a matrix of the access counters.

Figure 4 shows the percentage of the LRU blocks that are already dead when another block in the same cache set is referenced (i.e., read or write). As shown in the figure, on average, 93.9% of the LRU blocks in each cache set are dead blocks, meaning they are not re-referenced until evicted from the LLC. For the gcc benchmark, 99.3% of the LRU blocks are dead.

In this paper, we exploit this observation to enhance the write performance of the STT-MRAM caches. To enable fast writes, a cache block is invalidated in advance if the block is predicted as a dead block. Since the dead blocks are not re-referenced in the future, invalidating them from the cache in advance will not impact the system performance.

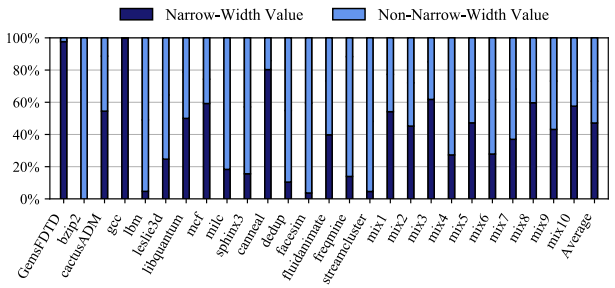



**FIGURE 6.** Proportion of narrow-width value and non-narrow-width value.

#### 3) SOFTWARE-LEVEL CHARACTERISTIC: NARROW-WIDTH VALUE

In many applications, operands of the arithmetic operations and the data stored in memory are usually much smaller than the full data width (i.e., 32 bits or 64 bits) of the processors. These small values are called narrow-width values. This well-known characteristic of the applications has been utilized frequently in many prior works for power, performance, and fault-tolerance optimizations [16], [18], [19], [31], [32]. In a narrow-width value, high-order bits (i.e., most significant bits) are all zeros. In this paper, we consider 64-bit word as the narrow-width value if its high-order 32 bits are all zeros.

Figure 5 shows a cache block (64 bytes) composed of eight words (8 bytes). Each word of a cache block can be either the narrow-width value or non-narrow-width value. For example, in the figure 5, the sixth word (denoted by W5) is the narrow-width value since its high-order 32 bits are all zeros while the third word (denoted by W2) is the non-narrow-width value.

Figure 6 shows the percentage of the narrow-width values among the data written to the LLC. On average, 47% of the data is the narrow-width value for SPEC CPU2006 benchmarks, as shown in the figure. Especially for some benchmarks such as GemsFDTD, gcc, and canneal benchmarks, the narrow-width values account for more than 80% of the data. Many prior works have shown similar observations. In [32], it is pointed out that more than 40% of the data is the narrow-width value. In [19], it is also pointed out that around 50% of the instructions use the narrow-width values as their operands.

The observation on the narrowness of data gives us an opportunity to reduce the write energy consumption of the STT-MRAM-based caches by skipping the state switching for the high-order 32 bits of the 64-bit word.

## III. PROPOSED CACHE MANAGEMENT TECHNIQUE

### A. OVERVIEW

The observations on the asymmetric write latencies of the STT-MRAM and the prevalence of the dead blocks motivate a novel cache management technique called **Proactive Invalidation (PROI)**. While a new block is installed in a cache set, PROI proactively invalidates a dead block from a cache entry of the same cache set and makes the entry as

the invalid state. At this time, all STT-MRAM cells of the cache entry are switched to the anti-parallel state as shown in Figure 7. Since all STT-MRAM cells of the proactively invalidated cache entries are in the anti-parallel state, the future writes on the cache entries will involve only the anti-parallel to parallel state transition, enabling fast writes in the STT-MRAM caches. In addition, even if PROI invalidates some blocks early from the cache, its performance impact can be trivial because it invalidates only the dead blocks that will not be re-referenced before being evicted from the cache.

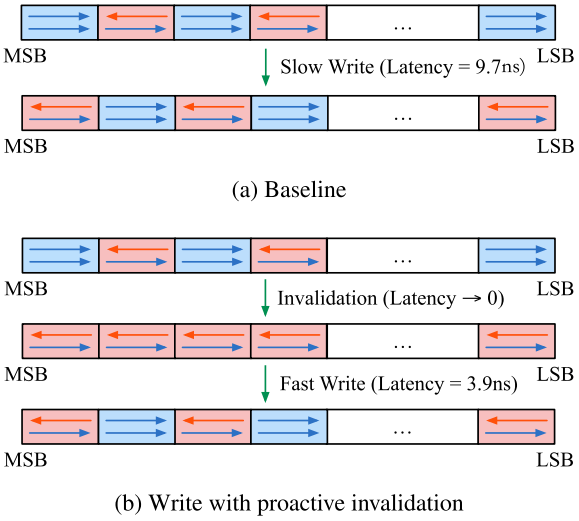



**FIGURE 7.** STT-MRAM write operations.

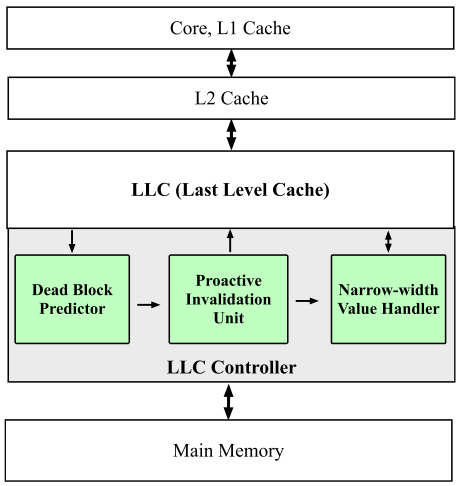



**FIGURE 8.** Memory hierarchy with PROI (Proactive Invalidation).

Figure 8 shows a memory hierarchy with PROI that employs additional three components: dead block predictor, proactive invalidation unit, and narrow-with value handler. Dead block predictor keeps track of the access status of the individual cache block to determine when each block becomes dead. When installing a new block in a cache set, the dead block predictor detects a dead block in the same set. The proactive invalidation unit sends an invalid signal to the cache to proactively invalidate the dead block. Finally, the narrow-with value handler prevents unnecessary bit flipping on high-order 32 bits by detecting the narrow-width values when performing the proactive invalidations or regular writes. In the following subsections, we will describe each component in more detail.

### B. DEAD BLOCK PREDICTION

PROI relies on the dead block prediction technique to select cache blocks that will be invalidated early from the LLC. If a dead block prediction is wrong, a live block can be evicted, increasing the miss rate of the LLC. Therefore, to avoid subsequent cache misses caused by the incorrect invalidations, PROI should employ an accurate dead block prediction technique. At the same time, we also consider the area and energy overheads involved with the dead block prediction. To meet this conflicting requirement, we propose a simple yet effective dead block prediction technique.

Our dead block predictor uses a saturating counter, called locality counter, per each cache set to monitor the hit rate of the individual cache set. The counter is initialized to its maximum value, incremented for a cache hit and decremented for a cache miss. While installing a new cache block in a cache set, if the counter value of the set is smaller than a threshold, the dead block predictor chooses one of the cache blocks in the set as the dead block. To choose the dead block, our simple predictor uses the replacement information. In the LRU (least-recently-used) replacement policy, the LRU block is evicted from the cache to make room for a new cache block. When the LRU block is evicted from the cache, the dead block predictor selects the second LRU block as the dead block.

In some cases, using the locality counter may fail to predict the presence of the dead blocks. For example, when only one cache block within a cache set is frequently accessed, the locality counter value will be greater than a threshold. In this case, a dead block that may exist in the set will not be invalidated. Even in this case, however, our dead block predictor will not reduce the potential performance gain of PROI. This is because it is not necessary to proactively invalidate a block for the cache set since new cache blocks will be rarely installed.

Figure 9 shows an example of the proposed dead block prediction and proactive invalidation. In this example, we assume that the STT-MRAM cache is a 4-way set-associative cache with four sets. We represent the recency of each block with the color brightness in the figure; the block with lighter red is more recently used than the block with a darker red. At time 0, only set 1 has four valid blocks: block A, B, C, and D. Initially, block A is the most-recently-used (MRU) block while block D is the least-recently-used (LRU) block in set 1. A read request for block E, which is also mapped to set 1, misses on the cache because the block does not exist in the cache at time 0. To accommodate block E, block D, which is the LRU block of set 1, is evicted from the cache at time 1. Once block D is evicted, block E is installed in the cache entry associated with the evicted block D (i.e.,

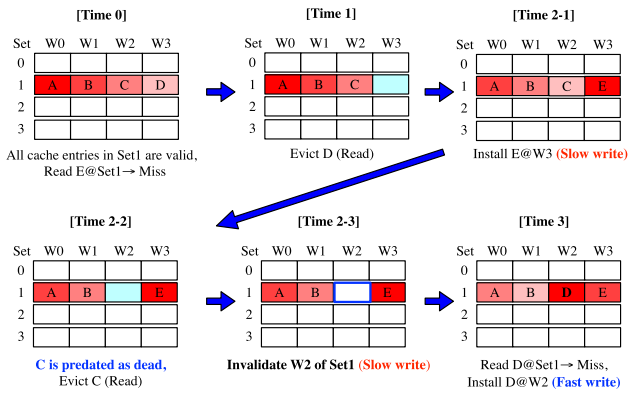



**FIGURE 9.** An example of dead block prediction and proactive invalidation. The operations at time 2-1, 2-2, and 2-3 are performed at the same time.

Way3 of the Set1). At the same time, the dead block predictor selects block C, which is the next LRU block, as a dead block and invalidates it in advance from the cache. While invalidating block C, all STT-MRAMs of the cache entry that holds the block is switched to the anti-parallel state. At time 3, another miss occurs while accessing block D, which is not in the cache and is also mapped to set 1. At this time, block D is installed with a shorter write latency than normal since the block is written to the entry that was proactively invalidated.

### C. PROACTIVE INVALIDATION (PROI)

In this subsection, we describe the proactive invalidation technique (PROI) in more detail. The primary goal of PROI is to shift the STT-MRAMs of cache entries to the anti-parallel state in advance so that the upcoming write operations on the invalidated cache entries involve only anti-parallel to parallel state transition. Since there will be no transitions from the parallel state to the anti-parallel state, which is much slower than the opposite direction transition, the write operations on the proactively invalidated cache entry can be performed in a much shorter latency than normal. As we discussed in the previous section, the write latency of the transition to the parallel state is 3.9 ns, while that of the transition to the anti-parallel state is 9.7 ns. So, the latency of a write operation performed on the invalid cache entry can be reduced to around 40% compared to the write operation on the valid entry.

PROI enables a fast write operation for the next cache fill operation by installing a new cache block into one of the proactively invalidated cache entries. Since the write operations installing new cache blocks account for more than 64% of the total writes in the LLC on average (Figure 10), reducing the write latency for the cache fill operations will have a high impact on the performance. Figure 11 shows a flow chart of the write operation with PROI. If the write operation is due to the cache fill that installs a new block in the cache, the cache controller searches for an invalid cache entry in the corresponding cache set. If there is an invalid entry, the new block is written to it, which can be performed with a short write latency (fast write, ❶). If there is no invalid entry in the

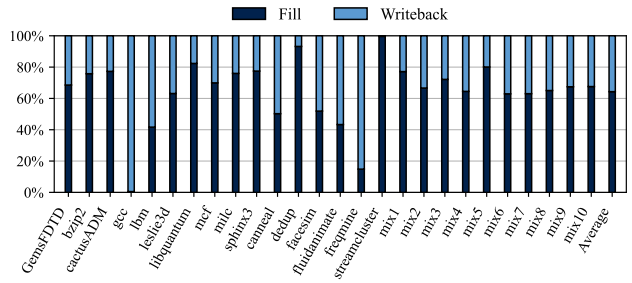



**FIGURE 10.** Breakdown of write operations in LLC.

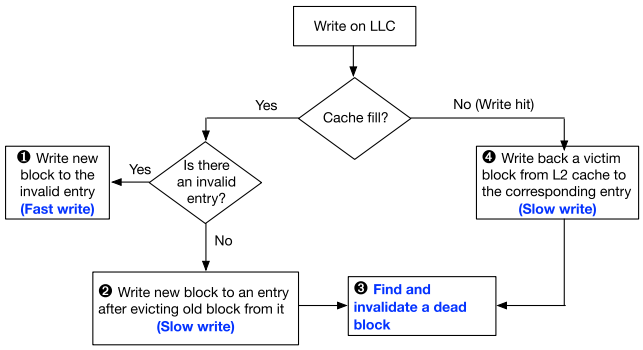



**FIGURE 11.** Flow chart of the write operation with Proactive Invalidation.

set, a victim block is evicted from the set, and the new block is written to the set. If it is the case, the write operation is performed with a normal write latency (slow write,❷). In the case of a write hit, the write operation is always performed with a normal latency (slow write, ❹). During the slow writes, a dead block is detected by the dead block predictor, and an invalid signal is sent to the cache I/O unit to invalid the dead block so that the corresponding cache entry becomes invalid (❸).

Even if PROI can reduce the average write latency, it can incur negative side effects if it is naively applied to the STT-MRAM caches. The first negative side effect is the additional write operations involved in the invalidation operations. The conventional cache uses a valid bit per each cache entry to indicate whether the content stored in the entry is valid or not. While a data block is written to a cache entry, the corresponding valid bit is set to 1. On the contrary, when invalidating the block from the cache, the valid bit is set to 0. Conventionally, the invalidation operation only updates the valid bit placed in the tag array. However, the proactive invalidation updates the contents stored in the cache entry as well as the valid bit to switch all STT-MRAM cells to the anti-parallel state. This will involve the additional write operations.

#### 1) INVALIDATE DURING WRITE

To hide the performance penalty due to the additional writes, the proactive invalidation is performed while a cache block is written with a normal write operation. In the conventional caches, write operations are involved when installing new

blocks to the cache (i.e., cache fill) or when writing back the dirty victim block from the upper-level cache to the current level (i.e., writeback). For both cases, PROI invalidates one of the dead blocks in the set where the normal write operation is performed. By performing the normal write operation and the invalidation simultaneously, we can hide the performance impact of the proactive invalidation.

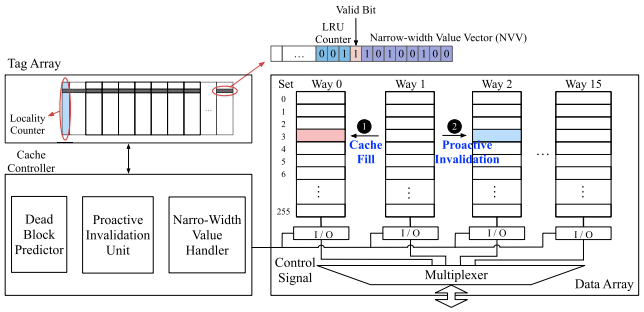



**FIGURE 12.** Cache Architecture with PROI. The tag array is extended to have locality counters used along with the LRU counter for dead block prediction. The NVV is used to indicate whether the corresponding words of a cache block are the narrow-width value or not.

Figure 12 shows a cache architecture with PROI. The tag array is extended to have a locality counter per each set. On cache access, all tag entries of a cache set are read to check whether the requested block resides in the set or not. At this time, the dead block detector selects a dead block by referring to the LRU counter fields of the cache entry and the locality counter of the set. When the locality counter value is lower than a threshold value, the LRU block of the set is selected as a dead block. Suppose it is the case and the current access involves a write operation (i.e., write hit or cache fill, ❶). In that case, the proactive invalidation unit generates an I/O control signal to invalidate the dead block from the set (❷). The proactive invalidation does not require any content written to the STT-MRAM cells. Thus, it can be implemented with a minor modification to the I/O circuits for injecting a write current with the same direction to all STT-MRAM cells.

#### 2) READ DURING INVALIDATION

The second negative effect is the additional read operations due to the invalidation of dirty blocks. For example, in Figure 9, block C is predicted as dead, and it is evicted from the cache. At this time, if block C is dirty, a read operation is involved in writing back the updated block C to the main memory. Due to this read operation, the proactive invalidation can increase the write latency of the STT-MRAM cache.

To hide the performance penalty due to the additional read operations, we exploit an attractive characteristic of the STT-MRAM cell. When performing a write operation on the STT-MRAM cell, the current state of the cell can be read in the early stage of the write operation [5]. This is because the read operation is the same as the write operation; both read and write operations are performed by injecting a current to the STT-MRAM. Furthermore, in a write operation, the free layer's direction is changed near the end of the operation [33], meaning that an STT-MRAM cell holds its previous value in the early stages of the writing operation. For example, the previous work demonstrates that a read operation can complete less than 1ns while a write operation consumes 10ns [5]. By exploiting this characteristic, a dirty dead block is read in the early stage of the invalidation. Then it is latched in the I/O circuit (i.e., the sense amplifier), hiding the latency for reading the dirty dead block from the critical path of the invalidation operation.

### D. PROACTIVE INVALIDATION-AWARE DATA ENCODING (PIDE)

Aforementioned above, PROI can efficiently reduce the write latency of the STT-MRAM caches. Unfortunately, however, the additional state transitions involved in the invalidation operation can increase the energy consumption. This subsection describes the first optimization called Proactive Invalidation-aware Data Encoding (PIDE) that reduces the additional state transitions caused by PROI.

Considering almost all modern high-performance processors are 64-bit architecture, the size of the word (i.e., the basic unit of data) is 64 bits. Meanwhile, many data processed by arithmetic units and stored in caches are small in many applications. Therefore, a large portion of data stored in a cache is a narrow-width value, as discussed in section II-C3.

To further study and confirm this characteristic, we observe the size of data stored in the LLC for SPEC CPU2006 and PARSEC benchmarks. As shown in Figure 6, almost all data in GemsFDTD, gcc, and canneal benchmarks are narrow-width values. On average, the parentage of the narrow-width values is around 47% across all benchmarks we observed. The prevalence of the narrow-width value intuitively indicates that the percentage of '0' bits in a cache is higher than '1' bits. Figure 13 shows the percentage of '0' and '1' bits in the LLC for SPEC CPU 2006 and PARSEC benchmarks. As shown, around 73% of bits stored in the LLC are '0' bit.

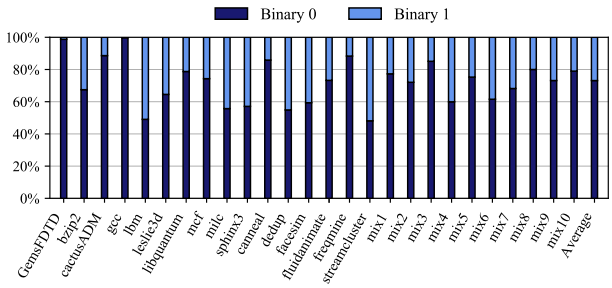



**FIGURE 13.** Percentage of "0" bits and "1" bits in the LLC.

This characteristic suggests an encoding scheme that uses the anti-parallel state to represent binary '0'. In many prior works [1], [34], [35], the parallel state is generally used to represent binary '0'. However, if we use this encoding scheme, there will be many bit flips from binary '0' to binary '1' in the invalidation operation because it switches all STT-MRAM cells to the anti-parallel state representing binary '1'.

Thus, to reduce the additional state transitions caused by the invalidation, it is more efficient to use the anti-parallel state to represent binary '0' and the parallel state to represent binary '1'. With this encoding scheme, we can reduce unnecessary state transitions (i.e., P → AP) in the invalidation operations since most STT-MRAM cells (i.e., around 73%) are already in the AP state representing binary '0'.

### E. NARROWNESS-AWARE PARTIAL WRITE (NPW)

This subsection describes the second optimization called Narrowness-aware Partial Write (NPW) that reduces the energy overhead of PROI by exploiting narrow-width values.

As described before, in the narrow-width values, the high-order 32 bits of the 64-bit word are zero. Therefore, if a significant amount of data stored in a cache are narrow-width values, the high-order 32 bits are rewritten to zero repeatedly. For example, in a typical program, an array with 64- or 32-bit data elements are initialized to zero, and the value in each element is accumulated within a for-loop. During the early iterations of the for-loop, the values are very small, meaning their high-order 32 or 16 bits are rewritten to zero repeatedly.

It is evident that repeatedly rewriting zero to the high-order bits is not necessary. The narrowness-aware Partial Write (NPW) technique employs a narrow-width value handler to avoid the unnecessary writes on the high-order bits in the invalidation and regular write operations. If a 64-bit word of a cache block is a narrow-width value, the high-order 32 STT-MRAM cells that hold the high-order 32 bits of the word are already in the anti-parallel state when PIDE is applied. Thus, if the 64-bit data stored in the cache is the narrow-width value, the narrow-width value handler does not inject the write current for the high-order 32 STT-MRAM cells when performing the proactive invalidations and regular writes. To this end, the access transistors of the high-order 32 cells and those of the low-order 32 cells are controlled independently with two individual local wordlines.

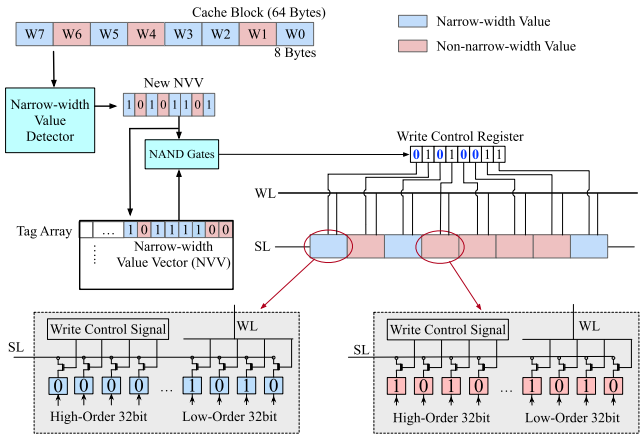



**FIGURE 14.** Implementation example of narrow-width value handler.

Figure 14 shows an implementation example of the narrow-width value hander that supports the narrowness-aware partial write. A 512-bit cache block is divided into eight 64-bit words. When the block is written to a cache entry, a narrow-width value detector checks whether each word of the block is narrow or not, and an 8-bit narrow-width value vector (NVV) is generated. In the example shown in Figure 14, the NVV of the new block is 10101101, meaning the first (W0), third (W2), fourth (W3), sixth (W5), and eighth (W7) words are the narrow-width value. The NVV of the new block that will be stored in a cache entry is compared with the NVV of the block currently stored in the cache entry. By comparing these two NVVs, an 8-bit write control signal is generated and latched in a write control register. Each bit of the control signal is used to control the access transistors of the high-order 32 STT-MRAM cells. If a bit of the control signal is 0, the corresponding access transistors are turned off. Each tag entry is extended to accommodate the NVV as shown in Figure 14.

**TABLE 2.** System configuration.

| | |
|---|---|
| **Processor** | 3.2 GHz, 4 Cores |
| **L1 Cache** | 32 KB, 8-Way, 4 Cycles, SRAM |
| **L2 Cache** | 256KB, 8-Way, 12 Cycles, SRAM |
| **Baseline Last-level Cache** | Shared 8 MB, 16-Way<br>STT-MRAM<br>Read Latency : 20 Cycles<br>Write Latency : 49 Cycles<br>Write Buffer : 8 Entries |
| **Last-level Cache with PROI** | Shared 8 MB, 16-Way<br>STT-MRAM<br>Read Latency : 21 Cycles<br>Fast Write Latency : 20 Cycles<br>Slow Write Latency : 49 Cycles<br>Write Buffer : 8 Entries<br>Locality Counter Threshold : 16 |

## IV. EVALUATION METHODOLOGY

To evaluate the performance and the energy efficiency of the proposed technique, we use gem5 simulator [36] in SE mode extended to model the STT-MRAM-based last-level cache (LLC) in the memory hierarchy. Table 2 lists the simulated system configuration. The level 1 (L1) and level 2 (L2) caches are assumed to be conventional SRAM-based caches, while the LLC is the STT-MRAM cache. The baseline LLC is configured to have 20 cycles of read latency and 49 cycles of write latency. In the LLC with PROI, the write latency is assumed to be 20 and 49 cycles for the fast and slow writes, respectively. Handling the narrow-width value is on the critical path of the read operation, and thus we conservatively assume that the read latency of the STT-MRAM cache with PROI is 1-cycle longer than the baseline.

PROI introduces some hardware overheads due to the additional components such as the Narrow-width value vector (NVV) and Locality counter. We use NVsim [37] to estimate the impact of the PROI on the design parameters of STT-MRAM caches such as area, per-access dynamic

**TABLE 3.** Design parameters of STT-MRAM caches.

| | Baseline | PROI |
|---|---|---|
| Area ($mm^2$) | 2.264 (1) | 2.347 (1.03) |
| Write Dynamic Energy (pJ) | 1076.61 (1) | 1108.16 (1.02) |
| Read Dynamic Energy (pJ) | 314.994 (1) | 346.545 (1.10) |
| Leakage Power (mW) | 71.242 (1) | 75.071 (1.05) |

*The numbers in parentheses refer to normalized values

**TABLE 4.** Workload mixes.

| | |
|---|---|
| mix1 | milc, sphinx3, GemsFDTD, libquantum |
| mix2 | GemsFDTD, leslie3d, cactusADM, libquantum |
| mix3 | milc, sphinx3, GemsFDTD, leslie3d |
| mix4 | sphinx3, lbm, mcf, milc |
| mix5 | libquantum, leslie3d, cactusADM, mcf |
| mix6 | mcf, lbm, sphinx3, leslie3d |
| mix7 | mcf, lbm, leslie3d, cactusADM |
| mix8 | bzip2, GemsFDTD, leslie3d, mcf |
| mix9 | mcf, sphinx3, cactusADM, bzip2 |
| mix10 | sphinx3, GemsFDTD, leslie3d, bzip2 |

energy (for read and write), and leakage power. Table 3 shows the STT-MRAM cache's design parameters obtained with NVsim modified to model the PROI. As shown in the Table, PROI increases the chip area, dynamic write energy, dynamic read energy, and leakage power of LLC by 3%, 2%, 10%, and 5%, respectively. These estimated parameters are used to evaluate the total energy consumption of the STT-MRAM caches in the system-level simulation with gem5. As will be described in Section V-D, even if PROI's overheads on the per-access read energy and leakage power are relatively high, their impact on the LLC's total energy consumption is insignificant. This is because the contribution of dynamic read energy to total energy consumption is small, and PROI reduces the execution time, resulting in less leakage energy consumption.

For evaluations, we use 16 memory-intensive workloads from the SPEC CPU2006 [20] and PARSEC [21]. We warm up the LLC for 10 billion instructions and perform the detailed simulation for 200 Million instructions. We execute the SPEC CPU2006 benchmarks in rate mode, where all cores run the same benchmark, and mix mode, where each core runs different benchmarks. As shown in Table 4, the SPEC CPU2006 benchmarks are randomly selected in the mix mode.

## V. SIMULATION RESULTS

### A. PERFORMANCE

Figure 15 shows the speedup of PROI when compared to a baseline configuration where the proactive invalidation is not applied. PROI achieves a speedup of 14% on average. For 17 of 26 workloads (65%), the speedup with PROI is more than 10%. The performance results show that the GemsFDTD, libquantum, and mix1 benchmarks benefit the most from PROI due to the dramatic reductions in the slow writes on the STT-MRAM-based LLC. These benchmarks suffer from frequent LLC misses as shown in Figure 16,

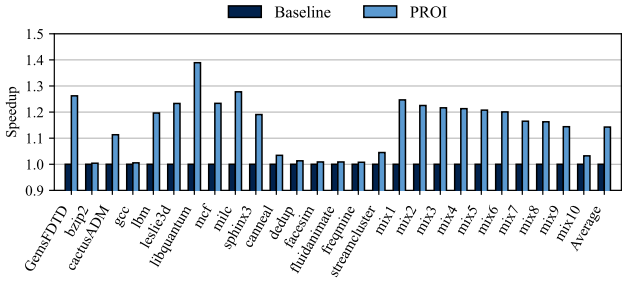



**FIGURE 15.** Performance.

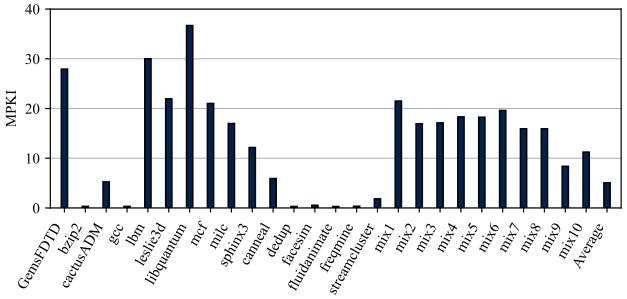



**FIGURE 16.** LLC MPKI (misses per kilo-instruction).

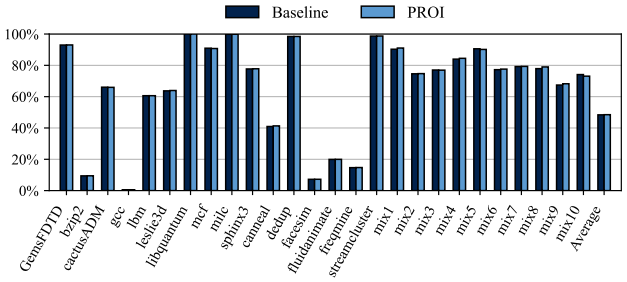



**FIGURE 17.** LLC Miss rate.

meaning there are many dead blocks in the cache, and new cache blocks are frequently installed into the cache. With PROI, the subsequent write operations involved in the block installation can be performed with a shorter write latency than normal (i.e., fast write). Thus, PROI reduces the average write latency significantly for the benchmarks with high MPKI (Misses per Kilo-Instructions), resulting in a significant speedup.

### B. MISS RATE

PROI evicts some blocks from the LLC in advance, and thus it can increase the miss rate. Figure 17 shows the impact of PROI on the miss rate of the LLC. As shown in the figure, even if PROI proactively invalidates some blocks from the cache, its impact on the miss rate is negligible. This is because the majority of LRU blocks are dead as shown in Figure 4 and accurately predicted as the dead block with our simple predictor. Since the dead blocks are not re-referenced in the future as described in section II-C2, proactively invalidating them from the cache does not increase the miss rate of the LLC.

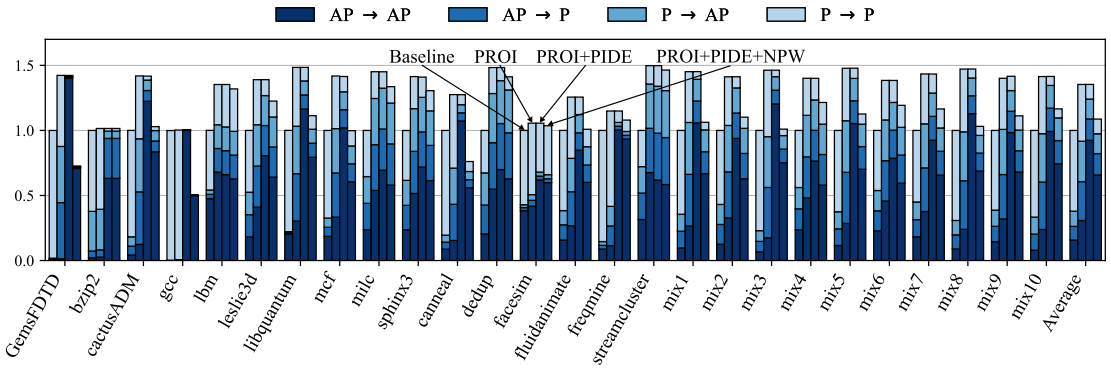



**FIGURE 18.** Breakdown of state transitions. The number of state transitions in PROI, PROI+PIDE and PROI+PIDE+NPW are normalized to the baseline.

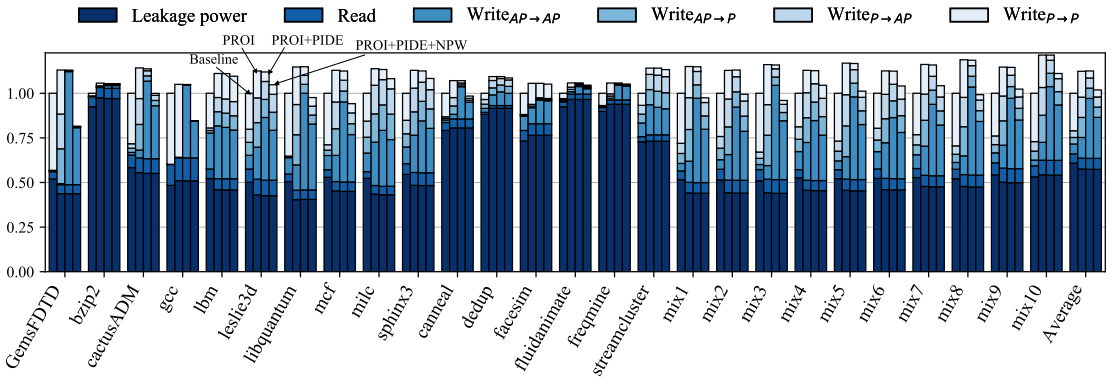



**FIGURE 19.** Breakdown of LLC energy consumption. LLC energy consumption with the proposed techniques is normalized to the baseline.

### C. STATE TRANSITION ANALYSIS

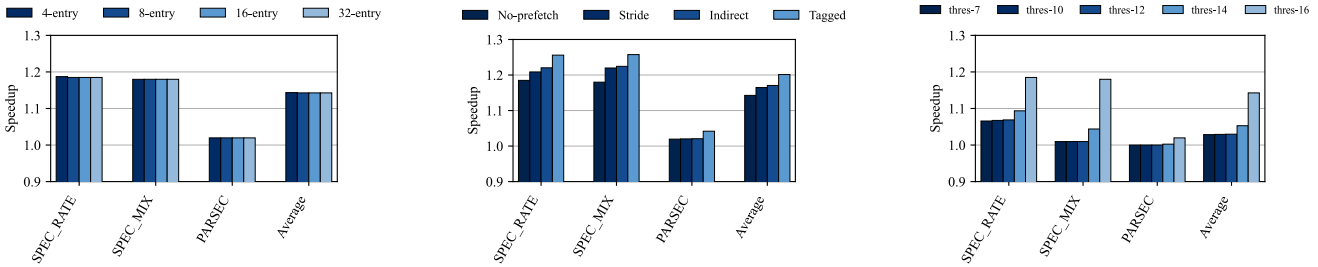
Figure 18 shows the breakdown of state transitions in the STT-MRAM caches with and without the proposed techniques. In the baseline, the parallel (P) to parallel (P) state transitions are dominant (62% on average). This is because a large portion of bits stored in the LLC is '0' bits, and the '0' bits are repeatedly written several time as we discussed in section III-D. When the PROI is applied, the total number of state transitions is increased compared to the baseline due to the additional write operations involved in the proactive invalidations. PROI switches the STT-MRAM cells to the AP state when it proactively invalidates the cache entries holding the dead block. Thus, the number of state transitions to the anti-parallel (AP) state (i.e., P $\rightarrow$ AP and AP $\rightarrow$ AP) are increased, especially for the benchmarks such as GemsFDTD, libquantum, mcf, and milc that have a high LLC miss rate. PROI also increases the numbers of state transitions to the P state from AP state. This is because the STT-MRAM cells in AP state (i.e., binary '1') of the invalidated cache entries are frequently switched to P state (i.e., binary '0').

The additional state transitions due to proactive invalidations can increase the energy consumption, offsetting the performance benefits of the PROI. To address this limitation, we can use the PIDE and NPW techniques. When the PROI is applied with PIDE, the AP to AP state transitions become dominant because the PIDE uses the AP state to represent binary "0". Since many bits stored in a cache are "0" bit, the proactive invalidation increases the AP to AP transitions which can be reduced by using the NPW.

When applying NPW and PIDE, the AP to AP state transitions are significantly reduced for all benchmarks. As shown in the figure, when NPW is enabled, the total number of state transitions decreases by 21% on average compared to the PROI+PIDE configuration. NPW reduces the AP to AP state transitions for the write operations involved in both regular writes (i.e., writeback and cache fill) and invalidations. Since many values stored in the cache are narrow-width, there are many AP to AP state transitions. Thus, NPW efficiently eliminates these unnecessary state transitions.

### D. ENERGY CONSUMPTION

Figure 19 shows the breakdown of LLC energy consumption. In the baseline, on average, the leakage energy accounts for 61%, the dynamic read energy accounts for 5%, and the dynamic write energy account for 34% of the total LLC energy consumption. The P to P state transition is dominant among the state transitions because '0' bits are frequently written to the cache as described in Section V-C. Thus its contribution to the total energy consumption is much higher than others (21% on average).




(a) The Depth of Write Buffer (b) Prefetcher (c) Locality Counter Threshold

**FIGURE 20.** Sensitivity to various parameters of LLC with PROI.

As discussed, the proactive invalidation increases the state transitions, increasing the LLC energy consumption significantly. With the PROI, LLC energy consumption is 12.3% higher than the baseline on average. As discussed in Section V-A, the PROI reduces the execution time, resulting in less leakage energy. However, it significantly increases dynamic write energy consumption due to the state transitions to AP state and the state transitions from AP to P states.

NPW with PIDE efficiently addresses the PROI's energy overhead. PIDE uses the AP state to represent binary "0". Since many bits stored in the cache are zero, PIDE converts the additional state transitions due to PROI into the AP to AP transition. As discussed, without any control mechanism, the AP to AP transition unnecessary consumes energy. NPW reduces these unnecessary energy consumptions by removing the state transitions for the narrow-width values in the write operations involved in invalidations and regular writes. As a result, when PROI is applied along with PIDE and NPW to the LLC, the LLC energy consumption is only 1.8% higher than the baseline.

### E. SENSITIVITY ANALYSIS

#### 1) IMPACT OF WRITE BUFFER

Many high-performance processors employ a write buffer to hold write requests, allowing the subsequent read requests to be serviced ahead of the preceding write requests. In such architecture, the impact of the long write latency on performance can depend on the depth of the write buffer.

Figure 20a shows the speedup of PROI over the baseline for different numbers of write buffer entries. As shown in the figure, PROI provides almost constant performance gain for the configurations with different write buffer entries. In the case of SPEC benchmarks running on rate mode (i.e., SPEC_RATE), PROI provides slightly higher performance for the write buffer with fewer entries. This is because the write latency can be more sensitive to the performance when there are small entries in the write buffer.

#### 2) IMPACT OF PREFETCHER

Figure 20b shows the speedup of PROI for the LLC with different prefetchers. PROI shows better performance than the baseline for all prefetchers. Prefetcher can increase the number of write operations in the LLC to install more blocks to it, making the LLC more sensitive to the write latency.

PROI's performance gain is higher when a more accurate prefetcher is employed for the LLC. This is because an accurate prefetcher reduces the miss rate of the LLC, and thus the write latency of the LLC becomes more critical to the performance. On average, PROI achieves a speedup of 16%, 16.5%, and 20% for the LLC configuration with Stride [38], Indirect [39], and Tagged [40] prefetchers, respectively.

#### 3) IMPACT OF LOCALITY COUNTER THRESHOLD

Our proposed dead block predictor determines whether a cache set contains a dead block by comparing a per-set locality counter to a threshold value. Thus, the effectiveness of PROI can be affected by the threshold value. As the threshold value increases, more sets are considered to have a dead block, invalidating the dead blocks more frequently. Figure 20c shows the effect of the locality counter's threshold value. As shown in the figure, PROI delivers better performance for the configurations with higher threshold values. This is because the majority of LRU blocks in a set are dead, as we described in Section II-C2. As a result, aggressive invalidations have a minor impact on the LLC miss rates while increasing the chances of having an invalidated entry in a cache set.

## VI. RELATED WORK

### A. DEAD BLOCK DETECTION

Several dead block predictors have been introduced in previous works and applied to address the cache efficiency issues. Tian *et al.* proposed a dead block predictor that samples the program counter to determine when a cache block is likely to be dead [26]. The access pattern is learned by a predictor and used for dead block replacement and bypass optimization. Chen *et al.* proposed an adaptive block placement and migration technique for an STT-MRAM-Based hybrid cache [41]. By identifying write patterns in the LLC, the proposed technique bypasses dead-on-arrival blocks to the main memory to reduce redundant write operations to the LLC. Similarly, Ahn *et al.* proposed a dead block-aware redundant write elimination technique for the STT-MRAM-based LLC. In [27], dead-block-aware prefetchers

were proposed by Lai *et al.* The proposed technique gathers traces of instruction addresses that access a specific block. If a trace returns the last access to one block, it will also return the last access to other blocks. Thus, the dead blocks are detected by indexing the sum of these instruction addresses.

### B. NARROW-WIDTH VALUE

Several researchers have attempted to take advantage of the properties of narrow-width values. For instance, a technique to enhance processor power and performance dynamically by using narrow-width operations and sub-word parallelism inside the core is proposed in [19]. Chen *et al.* [42] exploited the narrow-width values to optimize the main memory. Their proposed write technique detects whether a block is a ''Sparse Block'' (i.e., a block whose upper halves are all zeros). If so, the non-zero part of the block is used as a block's base to encode the block to reduce the write energy consumption and improve the performance. Hu *et al.* [43] proposed a technique to mitigate the soft error in register files, issue queues, and caches. When the data is narrow-width, its lower half is duplicated to the upper half. By comparing upper and lower parts when reading data from the storage elements, soft errors can be detected.

### C. ASYMMETRIC WRITE PROPERTY OF STT-MRAM

Bishnoi *et al.* proposed static and dynamic circuit-level approaches to decrease overall write latency in [4]. By boosting the write current exclusively for slow writes, the static approach reduces the write latency for the slowest transition (i.e., transition from P to AP). The dynamic approach gradually boots the write current to decrease the negative impact of the random write margin. Kim *et al.* [44] proposed two novel techniques to decrease the average written current of STT-MRAM: bit-line voltage clamping and 2T-1R dual-source line bit cell. The Bit-line voltage clamping technique can limit the current flow from the bit line to the source line by replacing the analog multiplexer with a pass transistor. This allows a bit cell to be clamped to a lower voltage when transitioning from the P to AP. Lee *et al.* [45] proposed a bit-line and word-line biasing technique to eliminate the asymmetric write property of the STT-MRAM.

## VII. CONCLUSION

In this paper, we proposed a new cache management mechanism called Proactive Invalidation (PROI), which improves the performance of STT-MRAM caches by proactively invalidating dead blocks to enable fast write operations. After combining two optimization techniques such as Proactive Invalidation-Aware Data Encoding (PIDE) and Narrowness-aware Partial Write (NPW), the Proactive Invalidation improves the performance by 14% with a small energy (1.8%) and area (3%) overheads. PROI provides an efficient framework for STT-MRAM caches by enabling fast writes and by eliminating unnecessary state transitions.

## REFERENCES

[1] Y. Chen, W.-F. Wong, H. Li, and C.-K. Koh, ''Processor caches built using multi-level spin-transfer torque RAM cells,'' in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Design*, Aug. 2011, pp. 73–78.
[2] S. Mittal and J. Vetter, ''A technique for improving lifetime of non-volatile caches using write-minimization,'' *J. Low Power Electron. Appl.*, vol. 6, no. 1, p. 1, Jan. 2016. [Online]. Available: https://www.mdpi.com/2079-9268/6/1/1
[3] R. Buhrman, ''Spin torque MRAM—Challenges and prospects,'' in *Proc. Device Res. Conf.*, 2009, p. 33.
[4] R. Bishnoi, M. Ebrahimi, F. Oboril, and M. B. Tahoori, ''Improving write performance for STT-MRAM,'' *IEEE Trans. Magn.*, vol. 52, no. 8, pp. 1–11, Aug. 2016.
[5] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, ''Energy reduction for STT-RAM using early write termination,'' in *IEEE/ACM Int. Conf. Comput.-Aided Design-Dig. Tech. Papers*, Nov. 2009, pp. 264–268.
[6] G. Sun, X. Dong, Y. Xie, J. Li, and Y. Chen, ''A novel architecture of the 3D stacked MRAM L2 cache for CMPs,'' in *Proc. IEEE 15th Int. Symp. High Perform. Comput. Archit.*, Feb. 2009, pp. 239–249.
[7] X. Wu, J. Li, L. Zhang, E. Speight, and Y. Xie, ''Power and performance of read-write aware hybrid caches with non-volatile memories,'' in *Proc. Design, Automat. Test Eur. Conf. Exhib.*, Apr. 2009, pp. 737–742.
[8] A. Jadidi, M. Arjomand, and H. Sarbazi-Azad, ''High-endurance and performance-efficient design of hybrid cache architectures through adaptive line replacement,'' in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Design*, Aug. 2011, pp. 79–84.
[9] Z. Wang, D. A. Jimenez, C. Xu, G. Sun, and Y. Xie, ''Adaptive placement and migration policy for an STT-RAM-based hybrid cache,'' in *Proc. IEEE 20th Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2014, pp. 13–24.
[10] Y.-T. Chen, J. Cong, H. Huang, B. Liu, C. Liu, M. Potkonjak, and G. Reinman, ''Dynamically reconfigurable hybrid cache: An energy-efficient last-level cache design,'' in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*, Mar. 2012, pp. 45–50.
[11] B. Kim, P. J. Nair, and S. Hong, ''ADAM: Adaptive block placement with metadata embedding for hybrid caches,'' in *Proc. IEEE 38th Int. Conf. Comput. Design (ICCD)*, Oct. 2020, pp. 421–424.
[12] K. Kuan and T. Adegbija, ''HALLS: An energy-efficient highly adaptable last level STT-RAM cache for multicore systems,'' *IEEE Trans. Comput.*, vol. 68, no. 11, pp. 1623–1634, Nov. 2019.
[13] F. Hameed, A. A. Khan, and J. Castrillon, ''Performance and energy-efficient design of STT-RAM last-level cache,'' *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 6, pp. 1059–1072, Jun. 2018.
[14] S. Hong, J. Lee, and S. Kim, ''Ternary cache: Three-valued MLC STT-RAM caches,'' in *Proc. IEEE 32nd Int. Conf. Comput. Design (ICCD)*, Oct. 2014, pp. 83–89.
[15] A. Ahari, M. Ebrahimi, F. Oboril, and M. Tahoori, ''Improving reliability, performance, and energy efficiency of STT-MRAM with dynamic write latency,'' in *Proc. 33rd IEEE Int. Conf. Comput. Design (ICCD)*, Oct. 2015, pp. 109–116.
[16] S. Hong and S. Kim, ''Lizard: Energy-efficient hard fault detection, diagnosis and isolation in the ALU,'' in *Proc. IEEE Int. Conf. Comput. Design*, Oct. 2010, pp. 342–349.
[17] S. Hong and S. Kim, ''A low-cost mechanism exploiting narrow-width values for tolerating hard faults in ALU,'' *IEEE Trans. Comput.*, vol. 64, no. 9, pp. 2433–2446, Sep. 2015.
[18] S. Hong and S. Kim, ''TEPS: Transient error protection utilizing sub-word parallelism,'' in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, May 2009, pp. 286–291.
[19] D. Brooks and M. Martonosi, ''Dynamically exploiting narrow width operands to improve processor power and performance,'' in *Proc. 5th Int. Symp. High-Perform. Comput. Archit.*, 1999, pp. 13–22.
[20] J. L. Henning, ''SPEC CPU2006 benchmark descriptions,'' *ACM SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 1–17, Sep. 2006, doi: 10.1145/1186736.1186737.
[21] C. Bienia, S. Kumar, J. P. Singh, and K. Li, ''The parsec benchmark suite: Characterization and architectural implications,'' Princeton Univ., Princeton, NJ, USA, Tech. Rep. TR-811-08, Jan. 2008.
[22] L. Liu, P. Chi, S. Li, Y. Cheng, and Y. Xie, ''Building energy-efficient multi-level cell STT-RAM caches with data compression,'' in *Proc. 22nd Asia South Pacific Design Automat. Conf. (ASP-DAC)*, Jan. 2017, pp. 751–756.

[23] M. Hosomi, H. Yamagishi, T. Yamamoto, K. Bessho, Y. Higo, K. Yamane, H. Yamada, M. Shoji, H. Hachino, C. Fukumoto, H. Nagao, and H. Kano, "A novel nonvolatile memory with spin torque transfer magnetization switching: Spin-RAM," in *IEDM Tech. Dig.*, Dec. 2005, pp. 459–462.

[24] Y. Zhang, X. Wang, Y. Li, A. K. Jones, and Y. Chen, "Asymmetry of MTJ switching and its implication to STT-RAM designs," in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*, Mar. 2012, pp. 1313–1318.

[25] T. Kawahara, K. Ito, R. Takemura, and H. Ohno, "Spin-transfer torque RAM technology: Review and prospect," *Microelectron. Rel.*, vol. 52, no. 4, pp. 613–627, Apr. 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S002627141100446X

[26] S. M. Khan, Y. Tian, and D. A. Jimenez, "Sampling dead block prediction for last-level caches," in *Proc. 43rd Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2010, pp. 175–186.

[27] A.-C. Lai, C. Fide, and B. Falsafi, "Dead-block prediction & dead-block correlating prefetchers," in *Proc. 28th Annu. Int. Symp. Comput. Archit. (ISCA)*. New York, NY, USA: Association for Computing Machinery, 2001, pp. 144–154, doi: 10.1145/379240.379259.

[28] H. Liu, M. Ferdman, J. Huh, and D. Burger, "Cache bursts: A new approach for eliminating dead blocks and increasing cache efficiency," in *Proc. 41st IEEE/ACM Int. Symp. Microarchitecture*, Nov. 2008, pp. 222–233.

[29] M. Kharbutli and Y. Solihin, "Counter-based cache replacement and bypassing algorithms," *IEEE Trans. Comput.*, vol. 57, no. 4, pp. 433–447, Feb. 2008.

[30] Z. Hu, S. Kaxiras, and M. Martonosi, "Timekeeping in the memory system: Predicting and optimizing memory behavior," in *Proc. 29th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2002, pp. 209–220.

[31] J. Hu, S. Wang, and S. G. Ziavras, "In-register duplication: Exploiting narrow-width value for improving register file reliability," in *Proc. Int. Conf. Dependable Syst. Netw. (DSN)*, 2006, pp. 281–290.

[32] G. Duan and S. Wang, "Exploiting narrow-width values for improving non-volatile cache lifetime," in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*, 2014, pp. 1–4.

[33] Y. Chen, X. Wang, H. Li, H. Liu, and D. V. Dimitrov, "Design margin exploration of spin-torque transfer RAM (SPRAM)," in *Proc. 9th Int. Symp. Quality Electron. Design (ISQED)*, Mar. 2008, pp. 684–690.

[34] X. Bi, M. Mao, D. Wang, and H. Li, "Unleashing the potential of MLC STT-RAM caches," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2013, pp. 429–436.

[35] X. Wu, J. Li, L. Zhang, E. Speight, and Y. Xie, "Power and performance of read-write aware hybrid caches with non-volatile memories," in *Proc. Design, Automat. Test Eur. Conf. Exhib.*, Apr. 2009, pp. 737–742.

[36] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, and R. Sen, "The gem5 simulator," *ACM SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, 2011.

[37] 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.

[38] T.-F. Chen and J.-L. Baer, "Effective hardware-based data prefetching for high-performance processors," *IEEE Trans. Comput.*, vol. 44, no. 5, pp. 609–623, May 1995.

[39] X. Yu, C. J. Hughes, N. Satish, and S. Devadas, "IMP: Indirect memory prefetcher," in *Proc. 48th Int. Symp. Microarchitecture*, Dec. 2015, pp. 178–190.

[40] A. J. Smith, "Cache memories," *ACM Comput. Surv.*, vol. 14, no. 3, pp. 473–530, 1982.

[41] Z. Wang, D. A. Jimenez, C. Xu, G. Sun, and Y. Xie, "Adaptive placement and migration policy for an STT-RAM-based hybrid cache," in *Proc. IEEE 20th Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2014, pp. 13–24.

[42] X. Chen, J. Wang, and J. Zhou, "Promoting MLC STT-RAM for the future persistent memory system," in *Proc. IEEE 15th Int. Conf. Dependable, Auton. Secure Comput., 15th Int. Conf. Pervasive Intell. Comput., 3rd Int. Conf. Big Data Intell. Comput. Cyber Sci. Technol. Congr. (DASC/PiCom/DataCom/CyberSciTech)*, Nov. 2017, pp. 1180–1185.

[43] O. Ergin, O. Unsal, X. Vera, and A. González, "Reducing soft errors through operand width aware policies," *IEEE Trans. Dependable Secure Comput.*, vol. 6, no. 3, pp. 217–230, Jul. 2009.

[44] Y. Kim, S. K. Gupta, S. P. Park, G. Panagopoulos, and K. Roy, "Write-optimized reliable design of STT MRAM," in *Proc. ACM/IEEE Int. Symp. Low Power Electron. Design (ISLPED)*. New York, NY, USA: Association for Computing Machinery, Jul. 2012, pp. 3–8, doi: 10.1145/2333660.2333664.

[45] D. Lee, S. K. Gupta, and K. Roy, "High-performance low-energy STT MRAM based on balanced write scheme," in *Proc. ACM/IEEE Int. Symp. Low Power Electron. Design (ISLPED)*, 2012, pp. 9–14.

**YONGJUN KIM** is currently pursuing the master's degree with the Department of Electrical and Computer Engineering, Sungkyunkwan University, South Korea. His current research interests include memory systems, non-volatile memory, and AI accelerator architecture.

**YUZE CHEN** is currently pursuing the master's degree with the School of Computer Science and Engineering at the Kyungpook National University, Daegu, South Korea. His current research interests include on-chip cache, non-volatile memory, and computer architecture.

**YONGHO LEE** is currently pursuing the master's degree with the Department of Electrical and Computer Engineering, Sungkyunkwan University, South Korea. His current research interests include heterogeneous memory systems, non-volatile memory, and computer architecture.

**LIMEI PENG** was an Assistant Professor at the Department of Industrial Engineering, Ajou University, Suwon, Republic of Korea, from 2014 to 2018. She was an Associate Professor at Soochow University, China, from 2011 to 2013. She is currently an Associate Professor at the School of Computer Science and Engineering, Kyungpook National University (KNU), Daegu, Republic of Korea. Her research interests include cloud computing, fog computing, data center networks, the IoT/IoV, 5G communications networks, and UAV communications.

**SEOKIN HONG** received the Ph.D. degree in computer science from the Korea Advanced Institute of Science and Technology (KAIST), South Korea, in 2015. From 2015 to 2017, he was a Senior Engineer at Samsung Electronics. During his two years there, he was involved in a project that developed the 3D-stacked memory. In 2017, he moved to the IBM Thomas J. Watson Research Center, where he worked on secure processor architectures and emerging memory/storage systems. He is currently an Assistant Professor at Sungkyunkwan University, South Korea. His current research interests include the design of low power, reliable, and high-performance processor architectures and memory systems. He received the Best Paper Awards from the International Conference on Computer Design (ICCD), in 2010 and the Design Automation and Test in Europe (DATE), in 2013.