TOWARDS HIGH-EFFICIENCY DATA MANAGEMENT IN THE NEXT-GENERATION PERSISTENT MEMORY SYSTEM

by

XUNCHAO CHEN
M.S. The University of Texas At Dallas, 2013
B.Eng. Nanjing University of Posts and Telecommunications, 2011

A dissertation submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy
in the Department of Electrical and Computer Engineering
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term
2017

Major Professor: Jun Wang

ABSTRACT

For the sake of higher cell density while achieving near-zero standby power, recent research progress in Magnetic Tunneling Junction (MTJ) devices has leveraged Multi-Level Cell (MLC) configurations of Spin-Transfer Torque Random Access Memory (STT-RAM). However, in order to mitigate the write disturbance in an MLC strategy, data stored in the soft bit must be restored back immediately after the hard bit switching is completed. Furthermore, as the result of MTJ feature size scaling, the soft bit can be expected to become disturbed by the read sensing current, thus requiring an immediate restore operation to ensure the data reliability. In this paper, we design and analyze a novel Adaptive Restore Scheme for Write Disturbance (ARS-WD) and Read Disturbance (ARS-RD), respectively. ARS-WD alleviates restoration overhead by intentionally overwriting soft bit lines which are less likely to be read. ARS-RD, on the other hand, aggregates the potential writes and restore the soft bit line at the time of its eviction from higher level cache. Both of these two schemes are based on a lightweight forecasting approach for the future read behavior of the cache block. Our experimental results show substantial reduction in soft bit line restore operations. Moreover, ARS promotes advantages of MLC to provide a preferable L2 design alternative in terms of energy, area and latency product compared to SLC STT-RAM alternatives. Whereas the popular Cell Split Mapping (CSM) for MLC STT-RAM leverages the inter-block nonuniform access frequency, the intra-block data access features remain untapped in the MLC design. Aiming to minimize the energy-hungry write request to Hard-Bit Line (HBL) and maximize the dynamic range in the advantageous Soft-Bit Line (SBL), an hybrid mapping strategy for MLC STT-RAM cache (Double-S) is advocated in the paper. Double-S couples the contemporary Cell-Split-Mapping with the novel Word-Split-Mapping (WSM). Sparse cache block detector and read depth based data allocation/ migration policy are proposed to release the full potential of Double-S.

In memory of my beloved grandmother.

ACKNOWLEDGMENTS

First, I would like to express my sincerest gratitude to my advisor Dr. Jun Wang for providing valuable resources and guidance to support my research. I feel very fortunate to work with Dr. Ronald DeMara and his Ph.D. student Dr. Navid Khoshavi. Without their inspiration, I could not complete my Ph.D. study and make this dissertation possible. I also would like to acknowledge pioneers in my research area, in particular, Dr. Yiran Chen from Duke University. Their previous researches build a solid foundation for my study.

Besides my advisor, I would like to thank my committee members, Dr. Mingjie Lin, Dr. Deliang Fan and Dr. Rickard Ewetz from Department of Electrical and Computer Engineering, Dr. Shaojie Zhang from Department of Computer Science, who have provided valuable suggestions in the preparation of this dissertation. In addition, I would express my appreciation to members from Computer Architecture and Storage System (CASS) Lab, specifically, Dan Huang, Xuhong Zhang, Jiangling Yin and Jian Zhou. Without their help and companion, I could not finish this dissertation that soon. Moreover, I would like to thank the Department of Electrical and Computer Engineering at University of Central Florida for the teaching assistantship. This dissertation is supported in part by the US National Science Foundation Grant CCF-1337244, 1527249 and 1717388.

Eventually, I would like to thank my parents for their unconditional and continuous support throughout my higher education journey.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

In order to meet the ever-growing demand for performance and energy efficiency, a large portion of modern processors is occupied by on-chip multi-level SRAM caches. However, the significant leakage power and cell area of SRAM impedes its future deployment in energy critical applications, such as mobile platforms. Despite several remarkable research advances to reduce the leakage power, it is inevitable to encounter additional SRAM energy consumption as CMOS technology continues to scale down. Recently, emerging non-volatile memory technologies, such as spintronic memories [68][21], have been identified as promising alternatives to SRAM. Among them, Spin-Transfer Torque RAM (STT-RAM) is considered to be an promising candidate for on-chip caches [30][40] due to its near-zero leakage and high cell density. In fact, companies like Qualcomm, already advanced their research in using STT-RAM to revamp the memory hierarchy [38].

By storing two or more bits in a single cell, Multi-Level Cell (MLC) designs boost data density and have been adopted in commercial products, such as MLC NAND flash. In light of this, MLC STT-RAM has been explored to enhance cache capacity [9][77]. Compared to the Single Level Cell (SLC) structure, one more Magnetic Tunnel Junction (MTJ) is placed into a single MLC STT-RAM cell. These two MTJs, whose feature sizes are maintained differentially to meet certain Tunneling Magneto-Resistance (TMR), can be implemented either in plane or perpendicularly. Various line-to-bit mapping strategies (e.g., direct mapping, cell split mapping) are enabled due to the distinct access characteristics of small and large MTJs in a MLC.

Despite the attractive features of MLC STT-RAM, reliability remains a current design issue [74]. There are three major challenges need to be overcame before deploying it as the last level cache or even persistent main memory. First is the write reliability issue, second is the read reliability issue, third is the bit mapping strategy. Accordingly, we propose three novel approaches in this paper as

shown in Fig. 1.1. (1) Design and implement an adaptive restore scheme which selectively over-write some soft-bit lines to reduce the overhead of write disturbance. (2) Design and implement an adaptive restore scheme which intentionally skip selected restoration to reduce the overhead of read disturbance. (3) Design and implement the Double-S mapping strategy which leverage the bit access feature to maximize the performance of MLC.

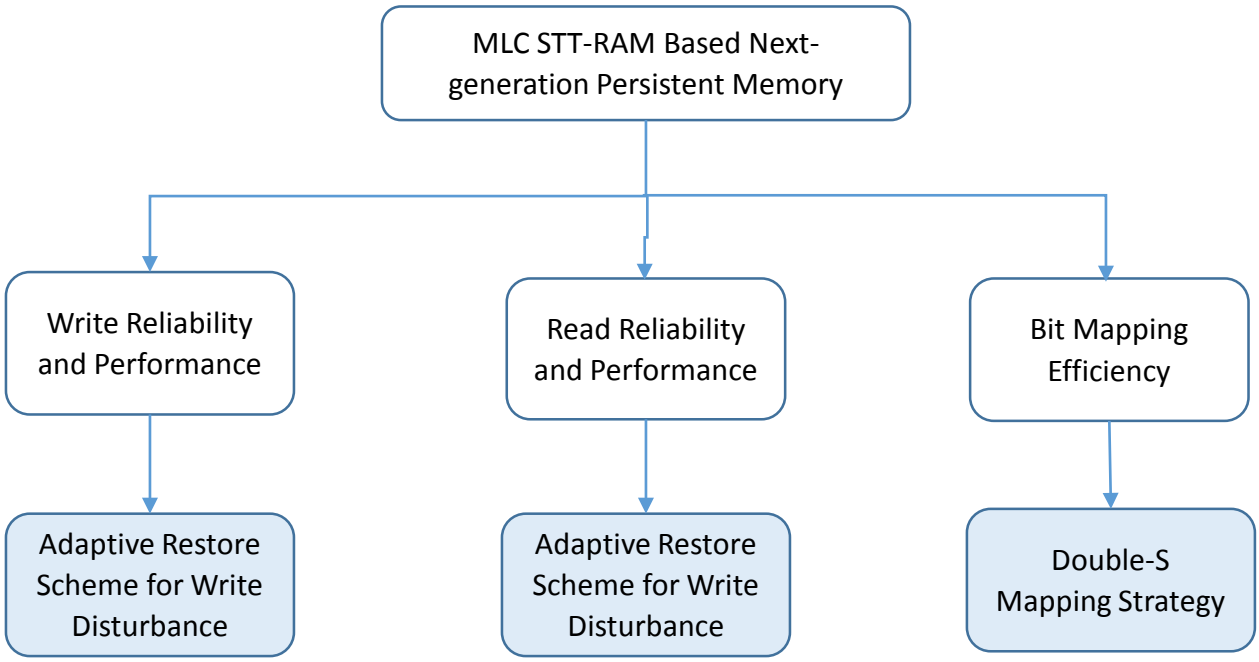

Figure 1.1: Research Work Overview

We will give a brief overview of each approach in the following sections.

1.1 Adaptive Restore Scheme For Write Disturbance

In the write operation of a MLC, the switching current for the large MTJ (hard bit) is able to change the magnetization direction of the corresponding small MTJ (soft bit), which is called

Write Disturbance (WD). To rectify WD, on every write access to a hard bit, data stored in the soft bit of the same MLC must be sensed out first and restored back after the writing is completed. Although a restore scheme guarantees data integrity, it introduces extra reads and writes and is energy inefficient in some scenarios. For instance, suppose a cache block is impacted by WD. In the case where that block is not subsequently read prior to its eviction, then its disturbed is insignificant.

Previous research on MLC STT-RAM mainly focused on leveraging the performance disparity of two MTJs. For instance, Jiang et al. [36] proposed to promote frequently written data into write-fast-read-slow soft bit lines, and frequently read data into read-fast-write-slow hard bit lines within parallel MLC structured STT-RAM. Although MLC doubles density, it also dramatically increases the write latency. As a result, this diminishes performance benefit gained from larger capacity. In [36], Line Pairing (LP) and Line Swapping (LS) are proposed to solve the latency limitation bottleneck so that the potential of MLC STT-RAM can unleashed. LP reconstructs two cache lines as a read fast write slow (RFWS) hard-bit cacheline and a read slow write fast (RSWF) hard-bit cacheline. The design promotes frequent write data into soft-bit lines and frequent read data into hard-bit lines. LS significantly reduces L2 the hit latency and enhance MLC performance.

Similarly, based on series MLC, Bi et al. [9] advocated a bit mapping strategy constructing general fast- and slow- lines which is referred as Cell Split Mapping (CSM). It takes only one sensing stage to read data out from soft-bit lines and writing it requires a small current which will not affect the hard bit lines. CSM leverages this feature of MLC and shares some similarities with LP. The major difference is CSM is based on series MLC which provides general fast- and slow- regions regardless write or read operations. By contrast, LP is proposed based on parallel MLC, which has write-fast-read-slow and read-fast-write-slow regions. Therefore, a simpler data migration strategy can be applied for CSM based MLC design to further increase the performance.

In this paper, we propose an energy efficient restore scheme, Adaptive Restore Scheme for Write Disturbance (ARS-WD), for MLC STT-RAM based cache. ARS-WD chooses to overwrite the soft bit line when it is less likely to be read in the near future. ARS-WD alleviates restoration overhead by intentionally overwriting soft bit lines which are less likely to be read. For the sake of this, first we define the concept of Read Reuse Distance (RRD), which is the timing distance between two consecutive read accesses. We also develop the RRD predictor inspired by an existing cache line reuse distance prediction design [39][55]. The RRD predictor samples memory access streams (hashed values of program counter) to calculate RRD at run time, and the RRD prediction table is updated for higher accuracy upon the incoming pairs of PC and RRD. Furthermore, to determine if it is harmless to overwrite, a threshold RRD with high coverage is adopted and compared with the Estimated Distance to the next Read (EDR). In the experiment part, we first compared the dynamic energy consumed by contemporary solution Immediate Restore Scheme for Write Disturbance (IRS-WD) with our proposed ARS-WD. Then we evaluate MLC with ARS-WD among four last level cache candidates. Our experimental results show ARS-WD fully release the potential of MLC to be the favored L2 alternative.

1.2 Adaptive Restore Scheme For Read Disturbance

On the other hand, in order to read data without disturbing the MTJ configuration, a small sensing current is applied to the bit line in MLC. However, with the continued scaling of MTJ feature size, the sensing margin narrows as the read current is mostly unchanged [52] while the switching current continues to be reduced. As a result, reading an MLC is likely to disturb the soft bits' stored value, which is commonly referred to as Read Disturbance (RD). Analogous to the solution in WD, an immediate restoration is required after every read of MLC. Again, this immediate restoration solution introduces extra reads and writes and is energy inefficient. For example, if a block is about

to be updated, immediately restoring it after RD turns to be a superfluous write operation.

Previous research on the read disturbance issue is mainly based on Single Level Cell (SLC) structure. For example, Mittal et al. proposed SHIELD, a technique that use data compression and selective duplication [51] to address the read disturbance issue and high write expense in terms of latency and energy. First, SHIELD avoids writing anything to all-zero data block because during the read stage of this block, all-zero can be reconstructed from compression encoding bits. SHIELD also duplicates the data block whose compressed width of at most 32B. During the next read, one copy of data is affected by RD while the other copy remains correct. The fact that many last level cache blocks are only read once can be leveraged to reduce restorations. For those data block with compressed width greater than 32B and uncompressed data item. a single copy of data is written and restored after each read.

In [35], the author studied the existing destructive and non-destructive read methods respectively. Destructive read means to write the sensed out data back into STT-RAM after each read so that data integrity can be guaranteed. Non-destructive read means with the help of redesigned sense amplifier, cell value can be sensed out with small enough read current yet longer duration. The author present two techniques to solve RD in SLC STT-RAM. First, Smash Read is proposed to trade read energy for shorter read latency when destructive high current restore required read is adopted to sense data. Second, Flexible Read is proposed to dynamically switch between different types of read schemes, Smash Read and low current long latency read, to maximize STT-RAM based main memory performance. Meanwhile, Flexible Read adaptively determine the destructive Smash Read or non-destructive low current long latency read to minimize the occupation time of memory. Both of these two research by Mittal et al. and Jiang et al. are based on SLC STT-RAM, while our approach tries to solve the read disturbance issue in MLC.

In this paper, we propose an energy efficient restore schemes for MLC STT-RAM based cache,

Adaptive Restore Scheme for RD (ARS-RD). Leveraging the non-inclusion property of modern multi-level cache hierarchy, and more importantly, the predictable access behavior of cache blocks, ARS-WD chooses to overwrite the soft bit line when it is less likely to be read in the near future. ARS-RD, on the other hand, postpones the disturbed block correction until the eviction of its most updated version from higher level. To enable ARS-RD, first we leverage the concept of RRD again, which is the timing distance between two consecutive read accesses to the same cache block. We also use the lightweight yet precise RRD predictor. In the experiment part, we first evaluate how much restore operations and dynamic energy can be reduce by ARS-RD compared to traditional Immediate Restore Scheme for Read Disturbance (IRS-RD). Then we evaluate comparison experiments among four last level cache candidates. Our comprehensive experimental results show ARS-RD and ARS-WD attains the full potential of MLC for L2 deployment.

1.3 Double-S Hybrid Mapping Strategy

Whereas previous research work [16][17][21][70][29][36] mainly deploy MLC STT-RAM as the on-chip cache, exploring this emerging NVM as the main memory is becoming an attractive topic [42][23]. However, there are several challenges need to be addressed in order to unleash the potential of MLC as the main memory. First, writes to the hard bit of MLC consumes significantly more energy and latency, how to minimize the impact of this becomes one of the priorities. Second, MLC STT-RAM can be deployed as memory extension collaborated with the DRAM memory, or DRAM memory replacement, storage cache lying between main memory and storage. The architecture of memory hierarchy also has a great impact on the overall system performance.

To address the first challenge, bit to cell mapping strategy has been a very promising yet not fully exploit direction. Wang et al. [70] proposed to dynamically disable the hard bits in cache line while keep the number of associativity. With the proposed interleaved mapping and block size re-

configuration strategies, only the soft-bits will be utilized for certain latency critical cache access. For normal cache accesses, both hard-bit and soft-bit line will be accessible to increase the cache hit with larger capacity. To address the second challenge, Kultursay et al. [42] evaluated SLC STT-RAM as a main memory alternative. Based on the observations that actual updated data in a memory row only takes a small portion and read can take more advantages of row buffer locality, the author propose two techniques to promote SLC STT-RAM as the main memory: tracking dirty blocks within rows for partial writes as well as writes bypassing the row buffer. Their experiment shows that with the optimization techniques proposed, STT-RAM main memory can save substantial power while maintain the same level of performance compared to DRAM main memory.

In this paper, we explore and evaluate MLC as the main memory. The goal of this research is to examine if MLC based main memory can either save power substantially or improve system IPC significantly. We evaluated the power and performance characteristics of MLC based main memory in details and conclude a hybrid memory architecture is necessary to exploit the advantages of MLC. We first introduce the unique challenges of employing MLC in the main memory system. Then we propose a novel mapping strategy Double-S, which leverages both intra and inter cache block access feature, to minimize the access and impact of hard bits. In the experiment part, we compare MLC enhanced by Double-S with DRAM as the main memory. We carefully evaluate the MLC based persistent memory and traditional DRAM based volatile memory in terms of energy consumption and system IPC. The result shows a hybrid memory extension architecture and Double-S will unleash the benefit of MLC.

The remainder of this paper is organized as follows. Chapter 2 presents the background of MLC STT-RAM cache. In Chapter 3, WD is analyzed first, then the overhead of the conventional restore scheme is assessed, then ARS-WD is presented. We also present the RRD prediction method and implementation in this chapter. Chapter 4 is organized in the same manner to present the design of ARS-RD. Extensive application-based experiments are conducted using the PARSEC benchmark

suite [10] to evaluate the proposed strategies. Chapter 5 explores challenges of employing MLC as the main memory and proposes an hybrid mapping strategy to maximize the use of advantageous soft bits.

CHAPTER 2: BACKGROUND

In this chapter, we introduce the basics of MTJ operation, Single Level Cell (SLC) STT-RAM, and Multi-Level Cell (MLC) STT-RAM. Two-step read and write schemes, and mapping strategies are then described for MLC designs. Lastly, the multi-level cache hierarchy with different inclusivity model are presented. We also review popular write policies including write-through with non-write allocation and write-back with write allocation. The trending best practice is deploying emerging non-volatile memory such as STT-RAM as part of the main memory. This enables persistent memory and makes it an interesting research topic. We discuss in this section how emerging NVM can cooperate with traditional DRAM memory, popular NVM and DRAM hybrid architecture.

2.1 Single Level Cell STT-RAM

2.1.1 Magnetic Tunnel Junction

STT-RAM is an emerging non-volatile memory which can provide SRAM-like read speed, DRAM-like density, and near-zero leakage power. Unlike traditional on-chip cache technologies using either latches (SRAM) or capacitors (eDRAM) as storage units, STT-RAM employs the non-volatile device — MTJ to store the data value. Each MTJ consists of two ferromagnetic layers (free and reference layer) and an oxide barrier (MgO) sandwiched between them.

The magnetization direction of the reference layer is fixed, while that of the free layer can be switched. MTJ resistance is determined by the relative magnetization direction of two ferromagnetic layers and thus it is programmable. As shown in Fig. 2.1(a) & Fig. 2.1(b), the magnetization directions of two ferromagnetic layers can be tuned to either parallel or anti-parallel, indicating whether the MTJ is in a low resistance state (logical 0) or a high resistance state (logical 1), re-

spectively.

2.1.2 1 Transistor 1 Junction Structure

Fig. 2.1(c) shows a popular '1T1J' structured STT-RAM cell where each MTJ is connected to a access transistor. In the write operation, a high voltage is applied between the source line (SL) and the bit line (BL), generating a current across the MTJ and switching the magnetization direction of the free layer. When reading data from a STT-RAM cell, a small sensing current is injected to generate a bit line voltage (V_{BL}). This V_{BL} is then compared with a reference voltage in order to decide whether a logical 1 or a logical 0 is stored in the cell.

Similar to other non-volatile memories, such as Phase Change Memory (PCM) and NAND Flash, write is more costly than read for STT-RAM in terms of latency and energy consumption. The expensive write operations also wear out MTJs gradually and thus wear leveling techniques are required to uniformly allocate the write. Moreover, the MTJ switching behavior is essentially an asymmetrical and stochastic process.

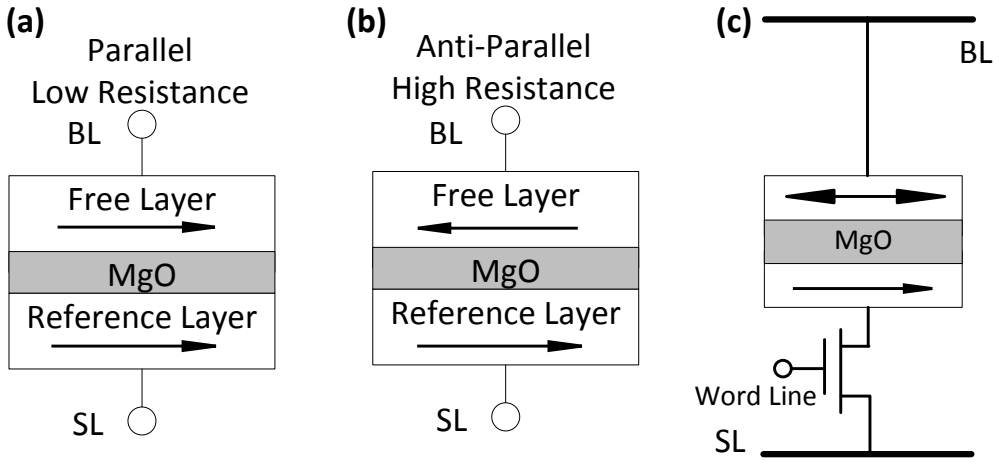

Figure 2.1: (a) Low Resistance State (b) High Resistance State (c) 1T1J STT-RAM Cell

2.2 Multi-level Cell STT-RAM

2.2.1 1 Transistor 2 Junctions Structure

To further improve the density of STT-RAM, MLC designs have been introduced and studied recently. There are two varieties of MLC structures, namely, serial MLC shown in Fig. 2.2(a) and parallel MLC shown in Fig. 2.2(b). The serial MLC stacks two MTJs vertically in a single memory cell and has been proven to be more reliable and easier to fabricate. On the other hand, the parallel MLC utilizes a single MTJ with a free layer having two independent fields, which leads to a smaller critical switching current compared to its serial counterpart.

We considered the serial MLC design in this paper since it is more practical and has been adopted in various implementations [32]. However, our proposed technique is also applicable to the parallel structure. In the serial MLC, two MTJs must occupy different cell areas so that four differential resistance states can be achieved. We refer to the small and large MTJs as the soft bit and hard bit storage respectively. Under constant resistance-area product, the soft bit, with larger resistance, is easier to be flipped than the hard bit because the soft bit requires a smaller switching current.

2.2.2 Read and Write Scheme of MLC

The read and write schemes of MLC STT-RAM have been well-studied in [20]. To read a 2-digit value from a MLC, it requires two comparisons as shown in Fig. 2.3(a). Recall that the sensing current passing through MTJs will produce a V_{BL}. This V_{BL} is compared with the reference voltage V_{ref1} first to decide the value of the soft bit, then a second comparison with V_{ref2} or V_{ref3} is done to decide the hard bit. With the knowledge of previous bit values stored in the MLC, resistance state transitions are depicted in Fig. 2.3(b) and described as follows:

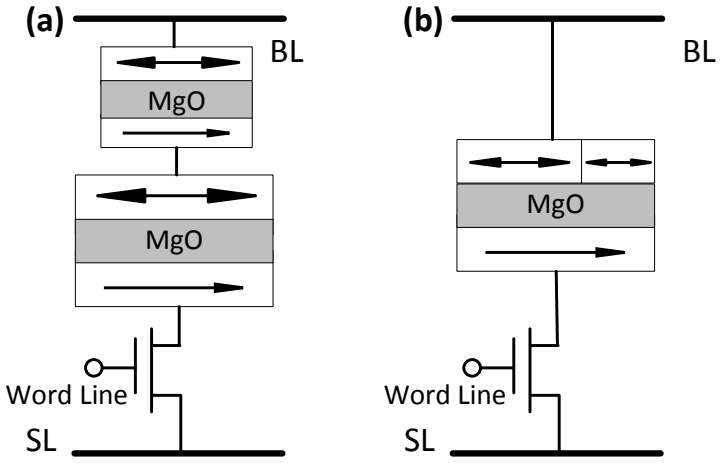

Figure 2.2: (a) Serial MLC (b) Parallel MLC

No Transition: MTJs remain at the original state when the incoming 2-digit value is identical to the previous one.

Soft Transition: A weak current I_{Low}, which will only affect the small MTJ, is applied to switch the soft bit only.

Hard Transition: If the hard bit needs to be changed and the soft bit needs to have the same value as the hard bit, then a strong current I_{High} is used to switch both MTJs.

Two-step Transition: When the hard bit needs to be flipped and the target 2-digit number has different values for both bits, then a hard transition is conducted first followed by a soft transition.

2.3 Logical Block to Physical Cell Mapping Strategies for MLC Design

2.3.1 Direct Mapping

Due to the distinct features of soft bits and hard bits, line-to-bit mapping strategies, such as direct mapping, cell split mapping, interleaved mapping [9][70] have been investigated to enhance the MLC cache performance. In direct mapping as shown in Fig. 2.4(a), although both bits in a MLC are mapped to the same cache line and thus it requires only one step to write the line, the fact that it is easier to access soft bits than hard bits is neglected. Thus, the access latency is always the worst-case because both soft and hard bits are sensed together. The hard-bit performance becomes the bottleneck.

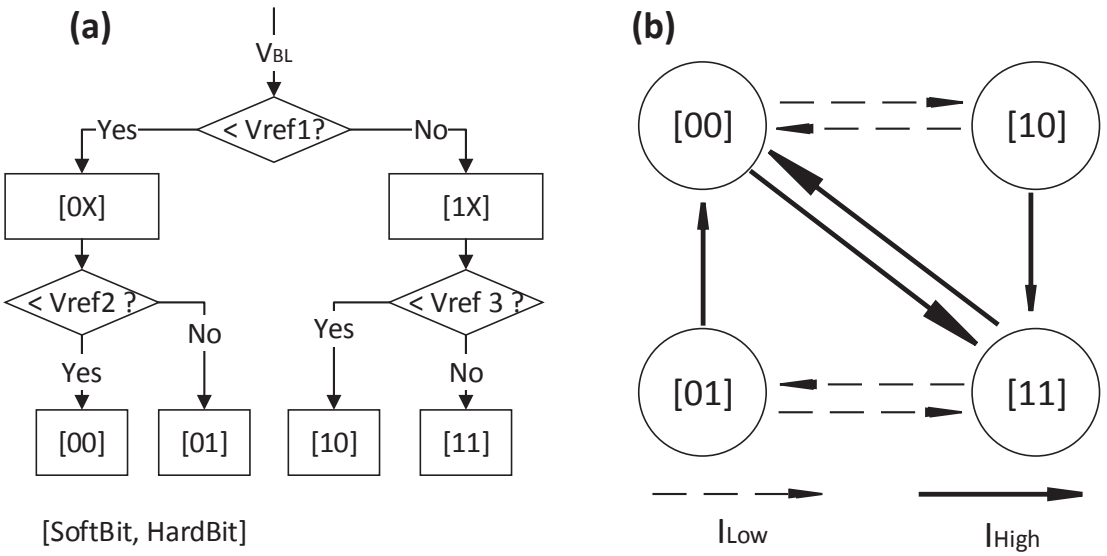

Figure 2.3: (a) Read Scheme in MLC (b) Write Scheme in MLC

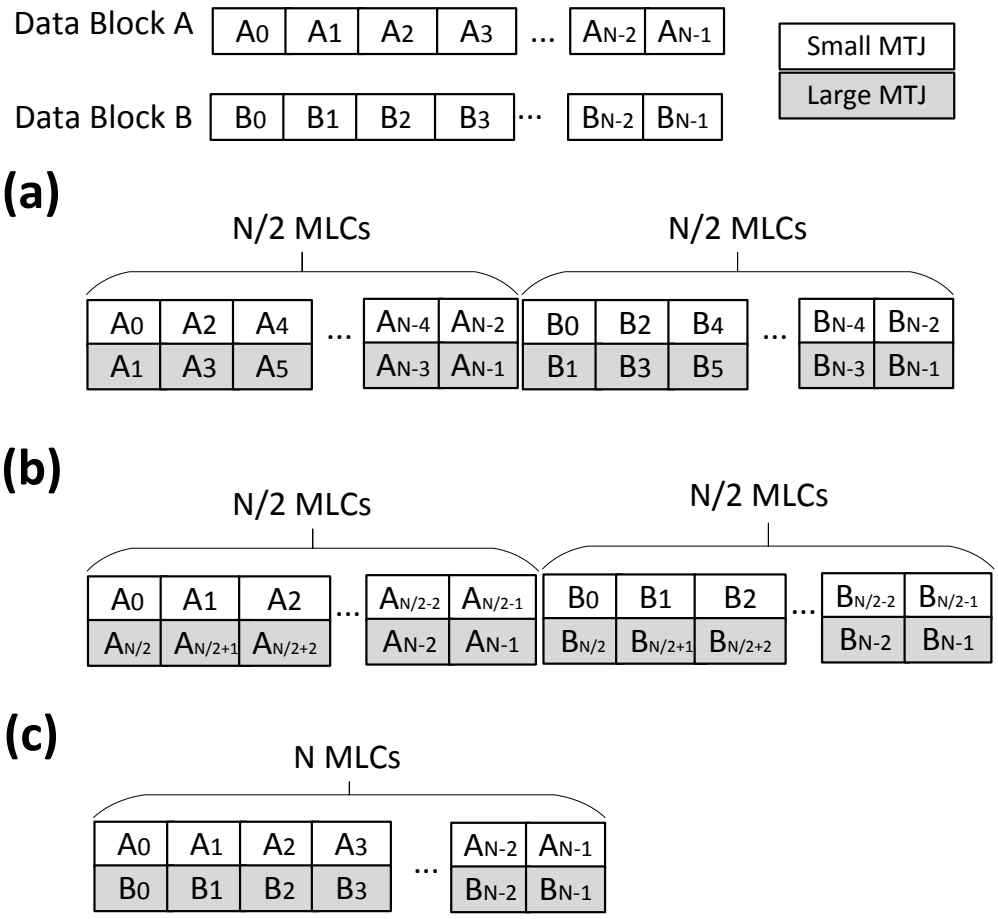

Figure 2.4: (a) Direct Mapping (b) Interleaved Mapping (c) Cell Split Mapping

2.3.2 Interleaved Mapping

Interleaved mapping leverages the non-uniform access frequency within a data block. As show in Fig. 2.4(b), the lower half of the data block is mapped to the soft-bit line of MLC, while the higher half is mapped to the hard-bit line. Interleaved mapping mitigates the write energy and performance for the lower half of the block since it takes only one-step to write for the soft-bit line. Obviously, there is no guarantee that the lower half will serve more write operations than the higher half. Thus, dynamic block size reconfiguration is required to support additional address

decomposition and mixed block modes.

2.3.3 Cell Split Mapping

Cell split mapping strategy constructs an entire cache line using the favorable soft bits. In other words, soft and hard bits in a MLC are mapped to two cache lines, referred as Soft Bit Line (SBL) and Hard Bit Line (HBL), respectively. As shown in Fig. 2.4(c), N MLCs construct two N-bit cache lines, where all the hard bits form an HBL and the corresponding soft bits in the same cells compose an SBL. By doing so, only one-step read and write are required to access the block in SBL. Furthermore, the fast and low-power feature of soft bits can be exploited by migrating the frequently accessed blocks to SBL.

2.4 Multi-level Cache Inclusion Properties and Write Policies

Different inclusion models (inclusive, non-inclusive and exclusive) can be employed for a multi-level cache. In this paper, we adopt non-inclusive cache hierarchy as industry has been trending towards it [1]. Also, non-inclusion property enables the write bypassing techniques to enhance the cache performance and energy efficiency [5][72].

2.4.1 Inclusive Model

In the inclusive model as shown in Fig. 2.5, the data in the higher cache level (L1 here) must be a subset of last level cache (L2 here). When there is a cache block evicted from LLC, the inclusion property is enforced by back invalidating the duplication of that data block (if any) in the smaller cache level. Therefore, the overall capacity of an inclusive cache organization is equal to

the capacity of LLC. The advantage of an inclusive model is it comes with snoop filter function naturally. When a data lookup in LLC leads to a cache miss, there is no need to send snoops to the core cache since the inclusion property guarantees the data not exist in the core cache. However, the back invalidation may cause serious performance degradation.

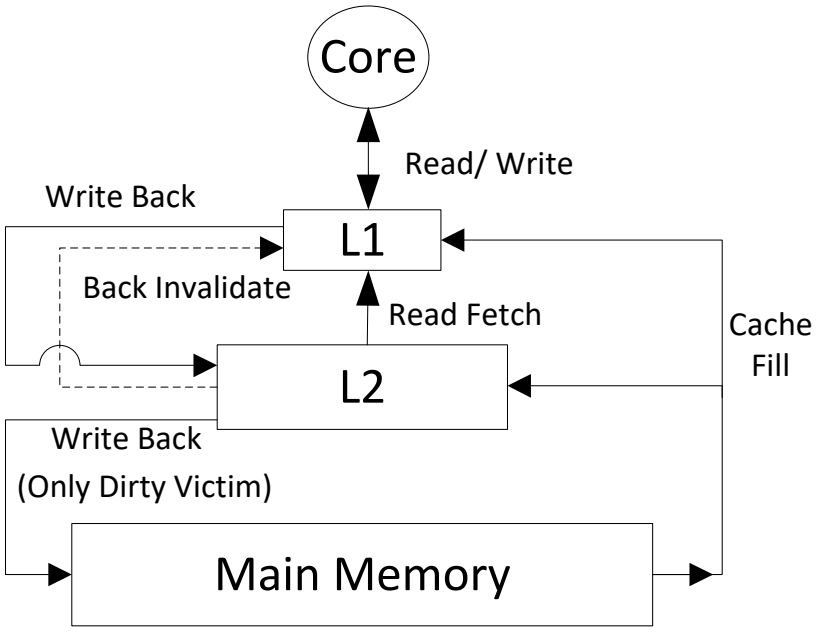

Figure 2.5: Inclusive Cache Hierarchy

For example, on a L2 miss, the missing data block is brought into both L2 and L1. While on a L2 hit, the block is brought into higher level cache and the duplicate copy stays in L2. On an L1 miss, one victim block will be selected for eviction. If it is a clean victim, it will be dropped directly. While if it is dirty, it will be inserted into L2. Whenever a data block is evicted from L2, the duplicated block in L1 is back-invalidated.

2.4.2 Non-inclusive Model

The non-inclusive cache hierarchy is showed in Fig. 2.6. Similar to inclusive cache, all the incoming data blocks are allocated to all levels of the hierarchy. The evictions of this model are silent and do not trigger back-validation. In other words, the non-inclusive model will not ensure the higher cache level is a subset of the LLC. As the result, the capacity of a cache hierarchy with non-inclusive property ranges from the sum of all levels to the size of LLC.

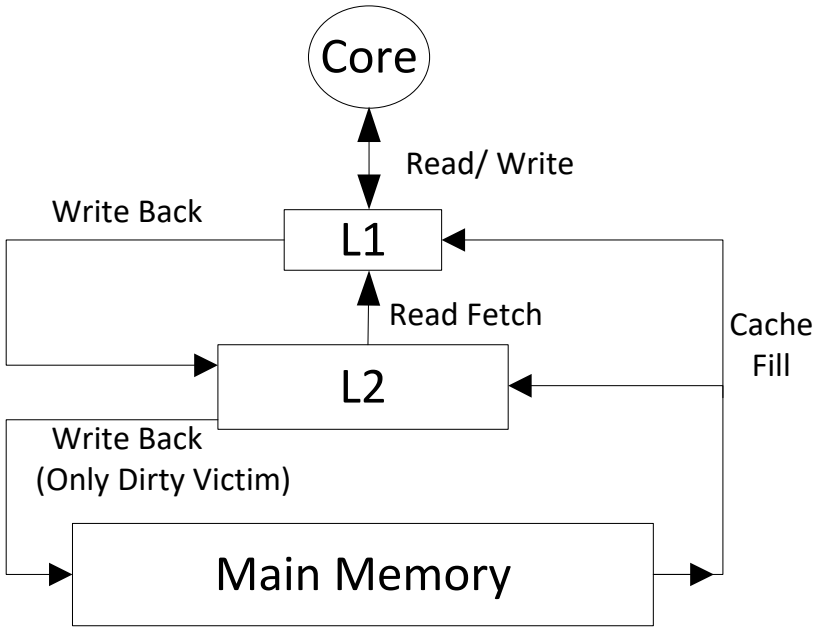

Figure 2.6: Non-inclusive Cache Hierarchy

Fig. 2.6 describes an non-inclusive two-level cache with write back policy. Triggered by a read miss on L2, the data block is fetched from the main memory and allocated to both L1 and L2, which is referred as a cache fill. On the other hand, when a block is evicted from L2 due to the replacement policy, there is no need to invalidate the victim block replica in L1, which is referred as back invalidation. In other words, compared to the inclusive cache, which guarantees L1 is a subset of L2, non-inclusive cache can accommodates substantially more data blocks occupying the

same physical space.

The capacity of an non-inclusive hierarchy is between the sum of all cache levels and the size of L2. Therefore, non-inclusive caches typically outperform inclusive caches by tradeoffs involving the snoop filtering effect [33]. When a block in L1 is selected for eviction, L1 tests its status first. If it is dirty, then this evicted block will be written back to L2. If it is clean, then this block will be overwritten by the incoming block without write-back.

2.4.3 Exclusive Model

In the exclusive cache organization as shown in Fig. 2.7, data block in any level is required not to have replica in any other level. The incoming data block from main memory will be filled into the highest level first. Upon the eviction from higher level cache, data block will be filled into the second smallest cache. In other words, all the stored data was previously evicted from upper level. Higher bandwidth is required for exclusive model because both clean and dirty victims will have to be written to the LLC from higher level. Compared to inclusive and non-inclusive models, the size of exclusive model is the largest. It equal to the sum of all caches in the organization.

For instance, on an L2 miss, the missing block is inserted into L1 only. While on an L2 hit, the data block is promoted to L1 and the copy in L2 is invalidated. Different from the non-inclusive model, a data block evicted from L1 will always be inserted into L2 regardless it is clean or dirty. Both non-inclusive and exclusive benefit from larger capacity while sacrifice snoop overhead.

2.4.4 Cache Inclusion Property Trade-offs

Trade-offs have been made between the inclusion property and various cache design constraints, such as capacity, coherence protocol complexity and write traffic density.

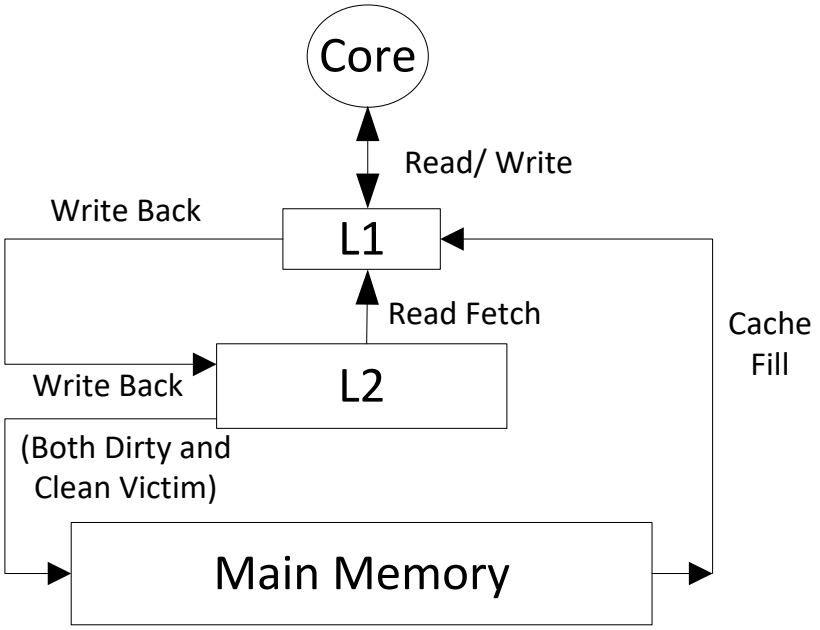

Figure 2.7: Exclusive Cache Hierarchy

Jaleel et al. [33] proposed Temporal Locality Aware (TLA) policies to achieve non-inclusive cache performance by mitigating the appearance of harmful inclusion victims. Jaleel et al. observed that the non-inclusive model benefits from not only the larger capacity, but also the reduction of inclusion victims. TLA eliminates the frequency of disadvantageous inclusive model victims. To enable TLA, three associated cache management policies are presented to identify cache blocks with high temporal locality (hot lines) and prevent the replacement of these hot lines in LLC until they become cold. The first technique is sending temporal locality hints to LLC so that the replacement policy in LLC is aware of the temporal information of a victim in the higher level cache. Second, the LLC intentionally selects a cache block before LRU policy and invalidates the block in higher level cache early. Upon a replacement, LLC queries the higher level cache for approval of selected victim. This query based selection technique takes advantage of the fact that core cache replacement stack has richer information about temporal locality. Various dead block bypass and placement policies have been presented to enhance the performance of non-inclusive

cache [5][72].

Sim et al. [65] presented a design adaptively selecting between exclusive and non-inclusive configuration based on the workload characteristics. Their proposal FLEXclusion can dynamically configures the cache inclusion model based on capacity requirement and traffic bandwidth. Because of FLEXclusion, LLC works in exclusive mode when the workload desires more cache capacity and in non-inclusive mode when extra capacity will not bring significant benefit so that bus bandwidth and energy dissipation can be saved. FLEXclusion requires some extra hardware to enable it, for example, workload behavior detector.

Recently, Cheng et al. [22] proposed a selective inclusion policy to reduce the write traffic, which considers asymmetric read/write properties of STT-RAM based LLC. Their experiment shows that none of inclusion models are very energy efficient. The lack of energy preferred inclusivity model issue still exists at different technology parameters. Even when the portion of read and write in workload varies, this lack of energy efficient inclusion presents in asymmetric read and write emerging NVM such as PCM, STT-RAM. The author also justify that FLEXclusion, which is designed for symmetric read and write memory technologies, will not be the optimal solution for NVM LLC. Motivated by these observations, the author proposed a new selective inclusion policy, loop-block-aware (LAP). LAP takes advantages of both non-inclusive and exclusive properties to cache a selective portion of higher level data so that the duplicated LLC writes can be reduced. LAP first identifies the clean data that is frequently reused and keeps a replica of this data in LLC to eliminate redundant insert operation. Meanwhile, LAP evicts cold clean data earlier in order to increase the free space in cache for use.

2.4.5 Cache Write Policies

Cache write policies have been well studied. There are two widely-used combination: write-through with no-write allocate and write-back with write allocate as shown in Fig. 2.8 and Fig. 2.9 respectively. When the CPU core issues write request to cache, the data will be eventually backup in low level. The timing of this write back is controlled by write policy.

In the write-through with no-write allocation, write request is completed synchronously to higher and lower level cache. Cache data block at the missed write address is not loaded to higher level, but write directly to the lower level memory. This is referred as the non-write allocation or write around. In this policy, only reads rather than writes are being cached. The subsequent writes will be directly written to the lower level. Therefore, they will not be beneficial to system and usually non-write allocation is combined with write through policy.

On the other hand, write back policy, also known as write behind, means writes are completed only at higher level cache. The write to the higher level is postponed until the data block in cache line is about to be updated or replaced. The write back policy usually come with write allocation, which is also referred as fetch on write. In this policy, data at the missed address is loaded to higher level cache, followed by a hit of write. Here write miss is treated similar to read miss.

From the implementation point of view, a write back cache with write allocation is more complicated. This is because it tracks which memory addresses have been over written, and set their status as dirty for backup data to the lower level. The cache block in these addresses are backup to lower level only when they are evicted from the cache, which is also referred as lazy write. Due to this, a read miss in this write policy requires two memory accesses, one is to replace the data from higher cache to lower level cache or memory, another one is promoted the requested data from lower level to higher level location.

2.5 Large Scale Persistent Main Memory

In the era of big data, the main memory plays a vital role to reduce the I/O overhead, in both High Performance Computer (HPC) cluster and commodity PC based data center. Big data programs can exhaust computer's DRAM based main memory space easily, especially for those iterative algorithms which generate tremendous intermediate results. To cope with the explosive growth in data volume and facilitate real-time analysis, many works have been done to either enlarge the memory space or treat the disk space as memory.

To enlarge the memory space, both scaled up and scaled out computers took advantage of the Moore's law for DRAM in early days. However, due to the dusk of this free lunch is approaching, researchers have been exploring new solutions to this problem. The limited memory capacity is typically addressed though the use of disk space. Swapping memory regions can cause data partition migration between disk and memory, which impedes the full potential of in-memory data processing engine, such as Spark.

In contrast, a multitude amount of recent work propose to perform big data analytics on computers equipped with Non-Volatile Memory (NVM) based persistent memory system, such as Phase Change Memory (PCM) or NAND flash Solid State Drives (SSDs). Unlike DRAM which uses capacitor to keep the data value and thus faces difficulty to keep the charge constant in size scaling, the emerging NVM leverages new material to meet the data retention requirement and can scale better than DRAM. Moreover, DRAM requires frequent cell refresh and causes large power consumption, which limits the amount of DRAM that can be deployed in the same package. NVM is non-volatile and does not require refresh power budget. Some NVM technologies even have comparable read performance to DRAM.

Most persistent applications come from storage level workloads such as file systems and databases,

where persistent memory can support crash consistency, in the other words, the persistence property. This persistence property makes sure that the metadata and other critical data stored in NVM remains a consistent state in the situation of program crash and power failure. For example, MLC STT-RAM is ideal for checkpointing as the checkpoint data can be stored in hard bit line while the normal data can be stored in soft bit line.

Non-volatile memory is typically byte addressable. Since NVM is good for scalability, the high density can enable NVM as an ideal candidate for large capacity persistent memory. However, emerging NVM such as PCM usually suffer from high write energy and high write latency compared to traditional DRAM technology. Therefore, hybrid memory architecture which takes advantages of both fast access DRAM and scalable NVM has been proposed in several research [26][60][76].

In addition, this emerging NVM is non-volatile. NVM outperforms traditional disk storage or last generation NVM such as NAND flash memory. Since data can be accessed in the same manner as in the main memory, NVM can be used as a persistent store. To advance the idea of hybrid memory, an unified memory architecture that uses NVM as both main memory and main storage has been proposed. In such architecture, files can be accessed with load and store instructions instead of block based I/O interfaces. The data can be moved between memory and disk without causing I/O bottleneck.

Also, NVM has been utilized as on-chip cache and disk cache. Deploying NVM as the last level cache can assist the system for higher density and lower power consumption. NVM can also be utilized as a file cache. In this case, the system performance and durability are both improved. In fact, NAND flash memory has been widely deployed in the contemporary massive storage system as disk cache or disk alternative.

NVM Solid State Drive (SSD) can be deployed in the memory hierarchy as system extension or

disk cache. In the first model, DRAM and SSD are managed as a single unified memory system. In the second model, SSD is utilized as part of disk. SDD lie under the standard block I/O interface and works as a cache for the disk. The inclusive property of multi-level cache, as we have described in the previous section, makes the first model superior to the second one in terms of increasing hit rate. However, the first model usually poses a modification requirement on operating system or even application. This brings significant challenges to researchers.

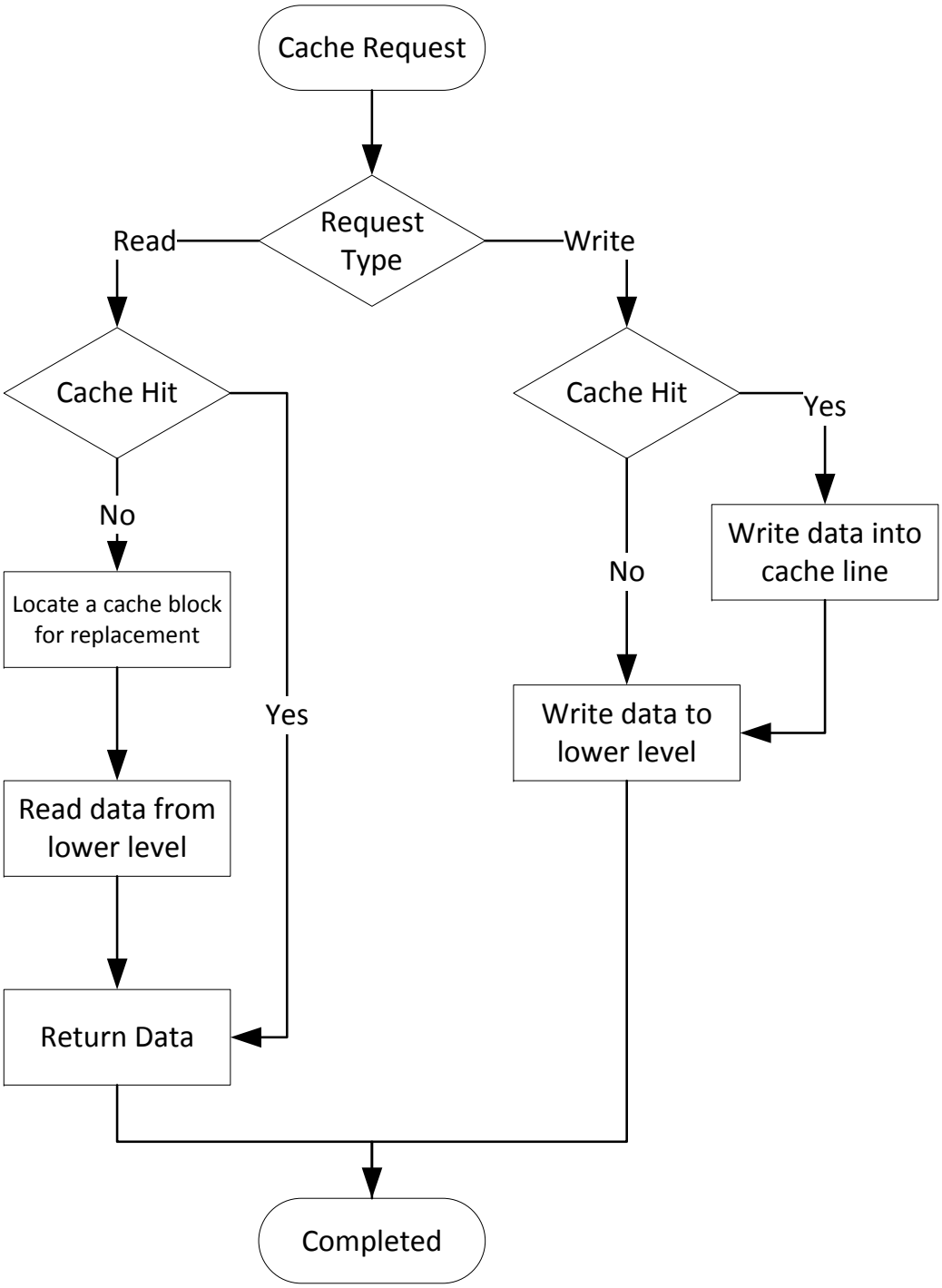

Figure 2.8: Write Through Cache with No Write Allocation

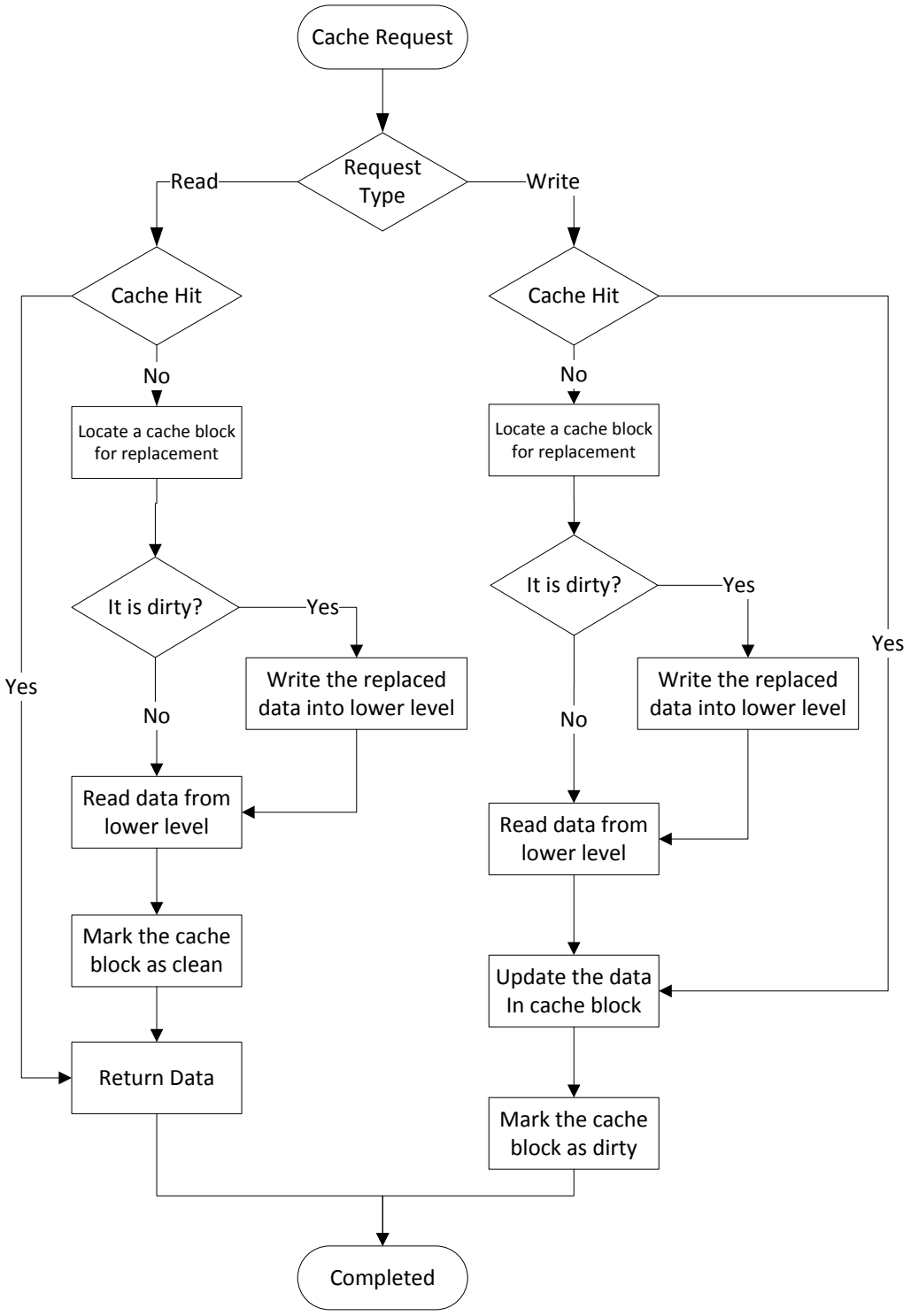

Figure 2.9: Write Back Cache with Write Allocation

CHAPTER 3: ADAPTIVE RESTORE SCHEMES FOR WRITE DISTURBANCE

3.1 Write Disturbance

In this section, we first elaborate the root causes of the inevitable write disturbances in MLC. We then demonstrate the significant energy overhead incurred by the naive Immediate Restore Scheme (IRS), which is adopted to overcome the disturbances. Finally, ARS-WD is proposed to mitigate the energy overhead incurred by restores. In order to enable ARS, the read reuse behavior of a block becomes critical. We will elaborate RRD and EDR in Section 5.1, and the adoption of threshold RRD value (RRD_{th}) in Section 5.3.

To facilitate our discussion, an 8-way associative cell split mapping cache set is depicted in Fig. 3.2. Here α_{HBL} is a cache data block stored in an HBL, β_{SBL} is another block saved in the corresponding SBL of α_{HBL}, both of which are in L2 of an non-inclusive cache hierarchy.

3.1.1 Motivation

As we described in Section 2, the switching current I_w flows through an MTJ and changes its magnetization direction. The write current value is proportional to its MTJ area [24] as defined in Equation 1:

$$I_w = A \cdot (J_{c0} + \frac{C}{T_\omega{}^\alpha}) \tag{3.1}$$

where J_{c0} is the critical current density at zero temperature; T_ω is the switching current duration; C and α denote fitting parameters. Whereas the feature size continues to shrink, the MTJ area will decrease exponentially. Therefore, the switching current amplitude continues to decrease as

technology scales to 22nm feature size and beyond as illustrated in Fig. 4.1.

In a serial MLC, switching the large MTJ requires higher current according to Equation 1, which will overwrite the value stored in the small MTJ. To rectify WD, upon each hard bit write request, data stored in the soft bit has to be read out first, then restored back immediately after the hard bit update is completed, which is also called Immediate Restore Scheme for WD (IRS-WD) as shown in Fig. 3.1. In other words, being unaware of the original value stored in the MLC, a two-step transition is always adopted in the hard bit write to ensure the data accuracy.

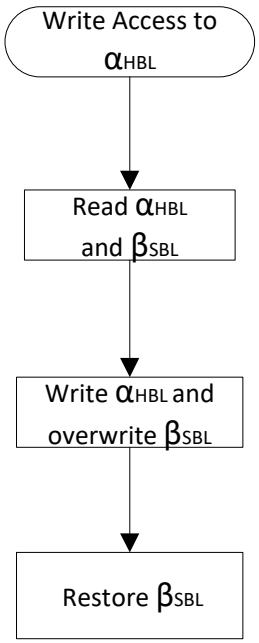

Figure 3.1: Immediate Restore Scheme for Write Disturbance

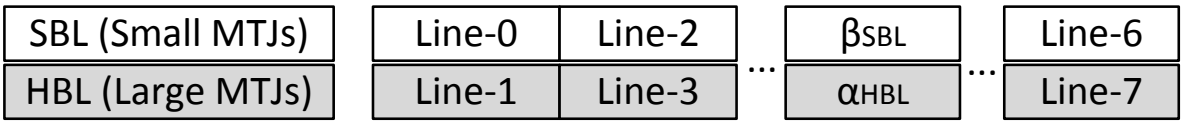

Figure 3.2: An 8-way Cell Split Mapping Cache Set

3.1.2 Energy Cost of Handling Write Disturbances

This inefficient, yet necessary IRS for correcting write disturbance will incur significant performance and energy consumption overheads. In particular, we illustrate the energy overhead caused by IRS on cell split mapping MLC STT-RAM cache. Assume E_{PC} is the dynamic energy consumed by cache peripheral circuitry, e.g. address decoder per access, E_{WSBL} and E_{RSBL} are the average write and read energy of SBL per request, while N_{WHBL} is the write access number of HBL. Upon every write to an HBL, the corresponding SBL has to be read (E_{RSBL}) and then become backed up in the write buffer first, then decode the address and write back (E_{PC} and E_{WSBL}) after HBL write is completed. The energy overhead spent on the IRS for WD is E_{IRS_WD}, modeled as below:

$$E_{IRS\_WD} = (E_{RSBL} + E_{PC} + E_{WSBL}) \cdot N_{WHBL} \tag{3.2}$$

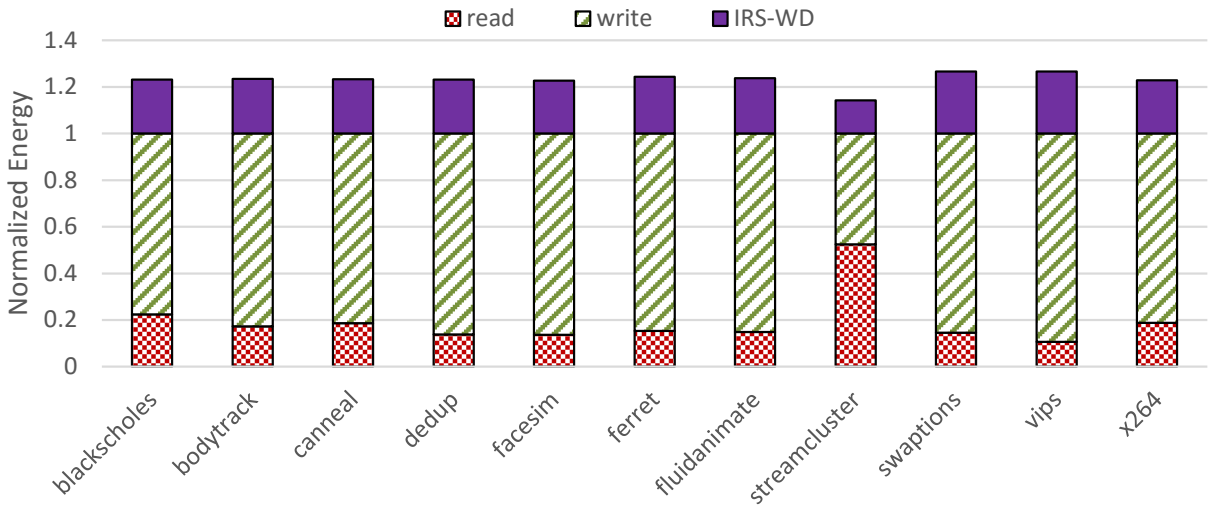

Figure 3.3: Dynamic Energy Breakdown of MLC STT-RAM with WD

Fig. 3.3 demonstrates the dynamic energy consumption breakdown of read, write and IRS for WD (IRS-WD) with PARSEC benchmarks (see Section 6.1 for detailed experiment settings). IRS-WD consumes 23% on average and up to 26.6% additional dynamic energy in write-intensive

workloads.

3.1.3 Adaptive Restore Scheme for Write Disturbance

The intuition behind MLC energy reduction is to avoid unnecessary restore operations. For example, in WD, immediately updating a block which will not be read soon is superfluous. In the Adaptive Restore Scheme for WD (ARS-WD) proposed in this section, by overwriting selected SBLs that are less likely to be read, a large portion of restorations can be reduced.

Recall the write scheme of MLC in Section 2.2, without the knowledge of the current value stored in a cache line, writing an HBL is a two-step transition. In contrast, ARS-WD selectively adopts one-step write, i.e. hard transition for HBLs, based on the read reuse feature of the corresponding SBLs. The flow chart of ARS-WD is shown in Fig. 3.4 and it works as follows:

- Upon a write request to α_{HBL}, L2 first checks the 'V' bit of β_{SBL}. If it is invalid ('V' bit is '0'), this line will only serve write access, but not read access. Overwriting it will not introduce any cache miss. The dirty bit of α_{HBL} is set when the write request is a dirty cache block write-back from L1. When the request is a new write allocation from the main memory due to L2 cache read miss causing cache fills, the dirty bit remains '0'.

- Next, ARS-WD detects if β_{SBL} has a replica in L1. If yes, then overwriting it will not incur an extra cache miss. Recall that in the non-inclusive hierarchy, upon a read miss in L2, the missing block is retrieved from the main memory to all cache levels.

  However, the copy stored in L2 will not be accessed until its eviction from L1. Thus, immediately restoring β_{SBL} will not benefit cache read hit. Also, since L1 maintains the most updated copy of the cache block, there is no need to write back β_{SBL} at the time of overwrite even if it is dirty.

- If β_{SBL} has no copy in L1, overwriting and invalidating this block may lead to a future read miss. Therefore, it is necessary to predict the read reuse behavior of β_{SBL} and calculate when the next time this block is read (EDR).

  Furthermore, we determine if β_{SBL} will be involved in subsequent reads shortly by comparing EDR with a threshold RRD value, which is adopted from the analysis on a wide variety of memory traces. When EDR is shorter than a threshold read reuse time (RRD_{th}), conventional IRS is applied to handle read miss.

- Supposing β_{SBL} has no copy in L1 (i.e., no write-back from L1) and will not be read in the near future according to the prediction result, thus no read fetch from L1, β_{SBL} can be overwritten in the write access of α_{HBL} without leading to more cache miss. Note here the dirty bit of β_{SBL} needs to be checked first. If it is '1', this block is written back to the main memory and is then overwritten.

Despite a confidence counter design is employed in RRD predictor as described in Section 5.1, misprediction can still occur occasionally. The worst case vulnerability occurs when an SBL is overwritten without restoration then read soon afterwards. In this scenario, an extra data fetched from the main memory is incurred, while data accuracy will not be affected as the SBL is invalidated first.

To leverage the advantages of SBL, various data migration policies have been proposed [9][73] which relocate frequently accessed cache blocks to SBLs. Although the data migration policy essentially alleviates the write pressure on HBLs, more than 20% of total writes are likely to be served by the unfavorable HBLs [73].

Moreover, due to the excessive write to SBLs after migrating cache blocks, SBLs wear out much faster than HBLs [49]. Therefore, wear leveling techniques will be required to prolong the lifetime

of MLC [21][49]. In this scenario, HBLs will serve even more writes to balance the MLC wear out.

In summary, ARS-WD is orthogonal to the existing data migration techniques, it becomes even more favorable when the wear leveling is applied.

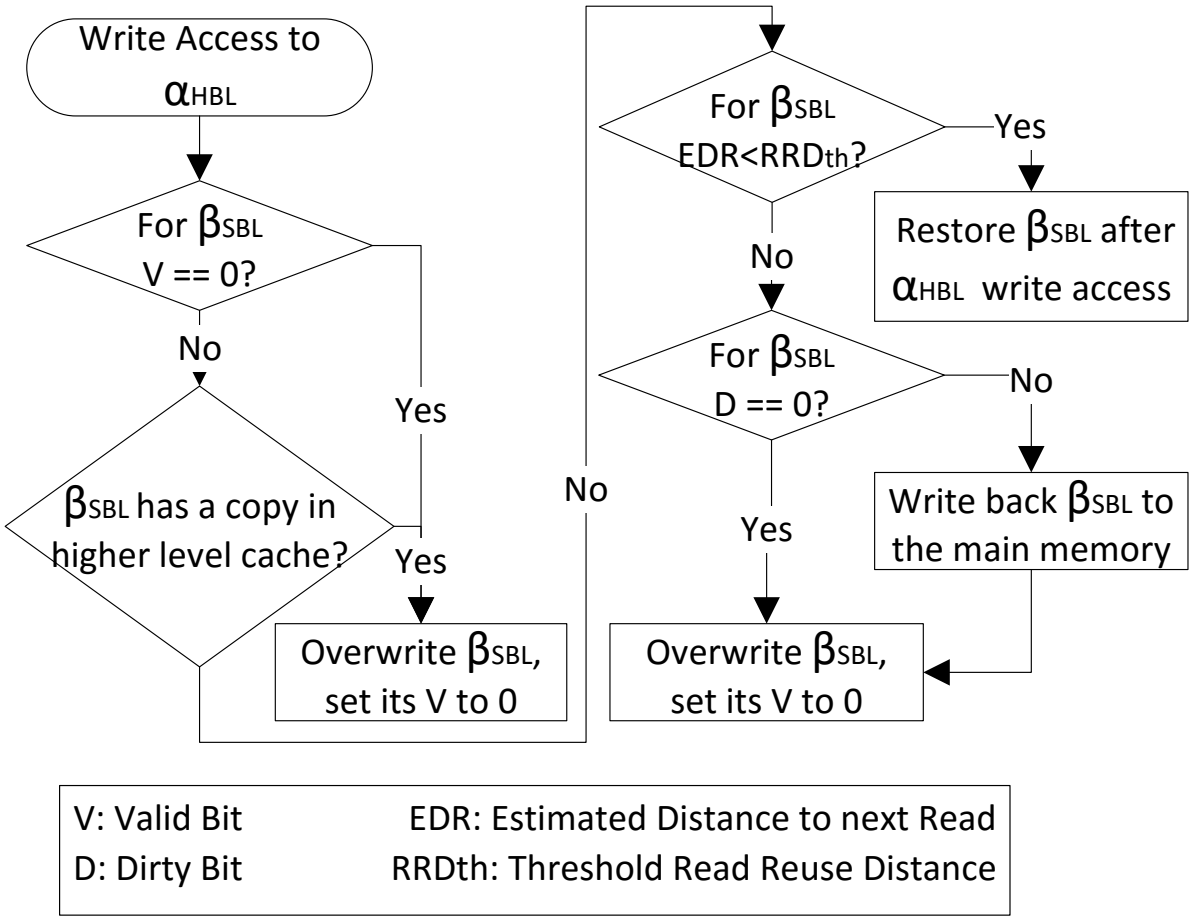

Figure 3.4: Adaptive Restore Scheme for Write Disturbance (ARS-WD)

3.2 Read Reuse Distance Prediction

In this section, we propose the concept of RRD and the associated PC-based predictor to enable ARS. Then the adoption of RRD threshold value and RRD predictor sampling period are presented based on comprehensive experiments. The details of experimental settings can be found in Section 6.1.

3.2.1 Prediction of Cache Block Behavior

On the other hand, there has been considerable research on cache block access behavior and its prediction. For example, Ahn et al. [6] proposed a write intensity predictor which identifies write intensive blocks and places them in SRAM portion of in a hybrid cache architecture. Khan et al. [62] presented a Read Reference Predictor (RRP) distinguishing between cache blocks that will be read and those that will not. RRP protects the performance-critical read reused cache block over the write-only ones.

Some cache management policies also take into consideration of the reuse distance of blocks to enhance cache performance [39][55][34]. However, none of these previous work predict the read reuse characteristics of cache blocks based on PC, and thus not completely address address the unique challenges brought by emerging on-chip cache technologies, such as MLC STT-RAM.

3.2.2 RRD Predictor Implementation

In contrast to the reuse distance measurement considering both read and write [34], RRD is the interval between two successive reads to the same block. In other words, RRD is a notion that quantifies data block read reuse frequency. Here, we use the number of intervening L2 access to

represent it. EDR is defined as the interval from the time an HBL is written to the next time this its corresponding SBL is read.

We illustrate the concept of RRD and EDR in Fig. 3.5. Here *timeline* describes the memory access stream, 'B' is a data block saved in an HBL being written at *Current_time*, while 'C' is the block stored in the corresponding SBL of 'B'. EDR is calculated using the last read access timestamp stored in the cache line, current time, and predicted RRD as below:

$$EDR = RRD - (Current\_time - Timestamp) \tag{3.3}$$

where Timestamp and Current_time are from the L2 access counter, and RRD is from the RRD predictor.

It has been observed that a specific instruction often executes a highly unique task (e.g., memory read access) and rarely changes this behavior. As instructions are uniquely associated with their Program Counters (PCs), which describes the instruction address in memory, PCs provides a very effective way of recording program context and predicting program behavior.

Previous work [39][55] proposed a PC-based reuse distance predictor to optimize cache replacement policy, by leveraging the fact that memory accesses can be grouped based on the instructions that caused them. In light of this design, we exploit instructions of cache accesses to predict block read reuse behavior. An alternate explanation for the validity of PC-based cache access behavior predictor can been found in related works such as [43][44][57], some of which utilize workloads similar to those used in this study.

Similar to prior work [39][56], there are two parts in our RRD predictor, namely, *read sampler* and *RRD prediction table*. Read sampler aims to calculate the corresponding RRD of a given PC. The sampling FIFO buffer is scanned for a matching PC on each read access. Simultaneously, read

sampler samples the access stream and stores into the sampling buffer. To calculate RRD, Read Sampler simply multiplies the sample period by its relative tail pointer position. For example, in Fig. 3.6, the sampling period is set to 2, therefore PCs associated with RdA, RdC are sampled into the buffer from access stream. WrtB is stored as a stall, i.e. a void state, regardless of its PC particularly. Then the WrtC request comes into cache followed by the RdC. WrtC is skipped in the PC matching process. While triggered by the next read access RdC, the sampling buffer is scanned for a match with the PC associated with block C (PC2). Since PC2 exists in the buffer already, a match is found and the RRD paired to this PC is computed as 4.

The second part of our RRD predictor, RRD prediction table, is similar to the one in [39], which is indexed by hashed PC and holds correlated RRD and Confidence Counter (CC). Up on every incoming pair of PC and RRD from the read sampler, the CC of the corresponding pair in RRD prediction table is updated. Only if the CC reaches a certain threshold, the RRD can be taken for prediction.

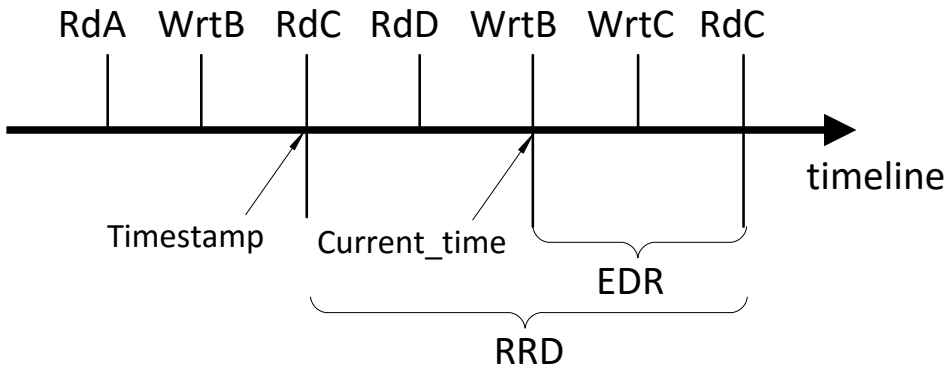

Figure 3.5: Read Reuse Distance and Estimated Distance to next Read

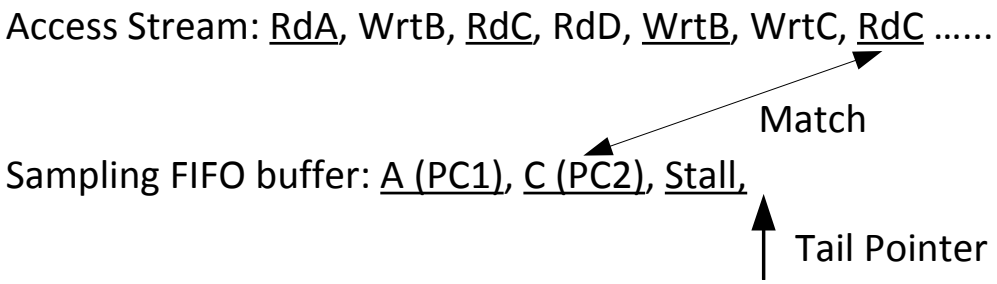

Figure 3.6: Read Sampler Operation

3.2.3 Novelty and Overhead of RRD Predictor

Our RRD predictor differs from previous designs in the following aspects. First, at the sampling stage, if a write request is sampled, a stall will be stored in the FIFO buffer as a placeholder without its PC. This is because we only care about read operations in the prediction. However, in [39], the PC associated with a sampled write request will be stored into the FIFO buffer for matching. Second, at the PC matching stage, only an incoming read request can trigger PC matching. This is due to ARS objective of estimating the read reuse distance instead of the general reuse distance in [39]. Lastly, in [39], a write back access filter is needed, which significantly increases the design overhead. This is because evictions from L1 appear as write accesses at L2. However, these write-backs are not associated with any instructions. Failure to consider this will substantially degrade prediction accuracy. Read Sampler explicitly avoids irrelevant PCs brought by evictions without the need for a write-back filter. Therefore, our predictor is more lightweight, but delivers even better performance over a range of workloads.

Regarding the memory size overhead incurred by RRD predictor, for example, a 512-entry RRD prediction table brings 512 entries $\times$ 39 bits (32 bits PC + 5 bits bucket + 2 bits confidence counter) per entry = 2.4 KB overhead, which is only 0.06% of a 4 MB cache. Also, for the 10-bit timestamp in each cache line, it only consumes 10 bits per 64 B = 1.95% of cache line size. According to the

circuit level simulation (see Section 6.1 for the setting), RRD predictor only contribute to less than 2% of energy consumption of cache line read, which is consistent with results from [72].

3.2.4 RRD Threshold Value

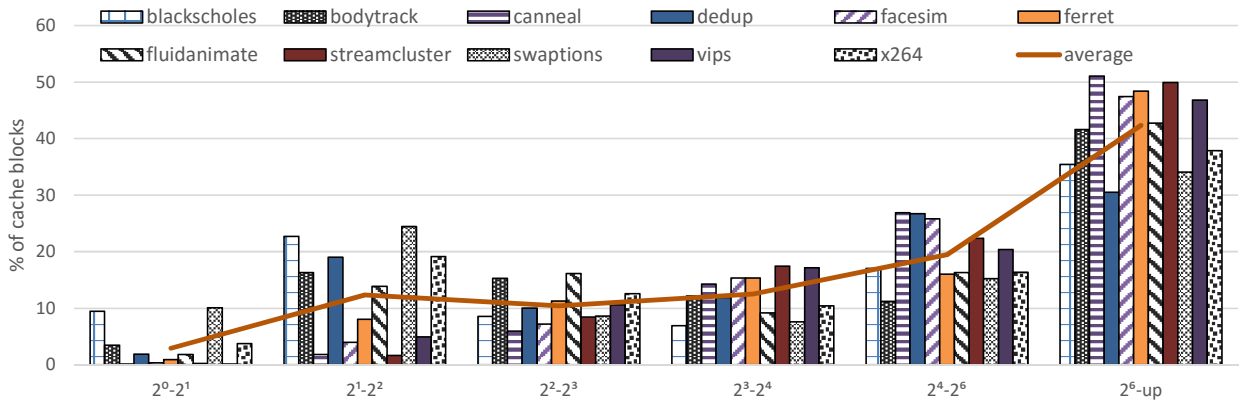

Figure 3.7: Percentage of total cache blocks associated with each RRD range

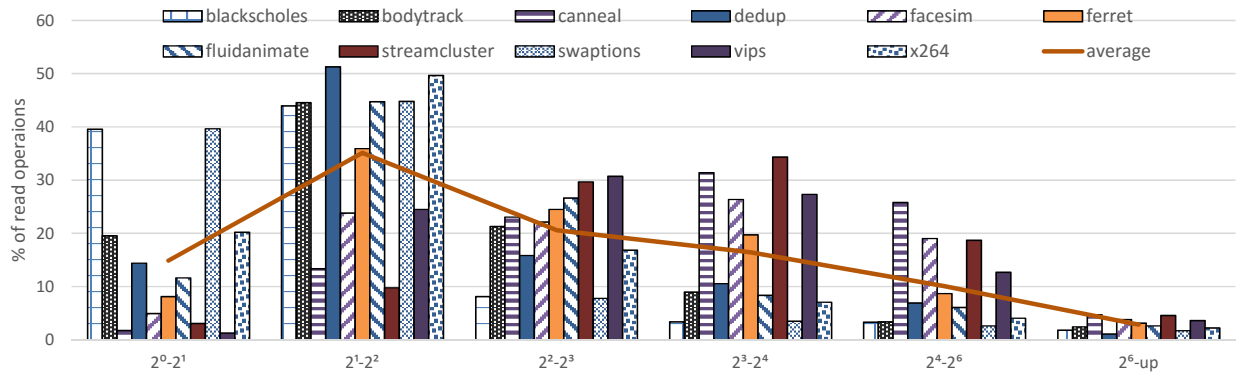

Figure 3.8: Percentage of total read operations associated with each RRD range

In order to determine whether restoration can be postponed or not, EDR of an SBL needs to be compared with a threshold RRD value. This value is adopted from the analysis of various memory accesses. Note here RRD value is presented by the power of two as this can help us define RRD

access counter size. Fig. 3.7 shows the percentage of cache block associated with each range of RRD. For example, cache blocks whose RRD is greater than 2^4 account for 62% of the total number on average. In other words, more than half (62%) of the cache blocks have a distant RRD and are eligible for overwrite if 2^4 is selected as the threshold. Moreover, the number of reads associated with these blocks takes only 13% of total read accesses as described in Fig. 3.8. That means overwriting these blocks is less likely to incur read misses. Our goal here is to find out a RRD range, where the associated cache block percentage is maximized so that more restoration can be skipped, while the number of read operation on these blocks are insignificant, i.e. the miss penalty increase is tolerable. Hence, we adopted 2^4 as the threshold RRD. Blocks with EDR greater than the threshold, i.e. non read-intensive, are selected to skip restorations.

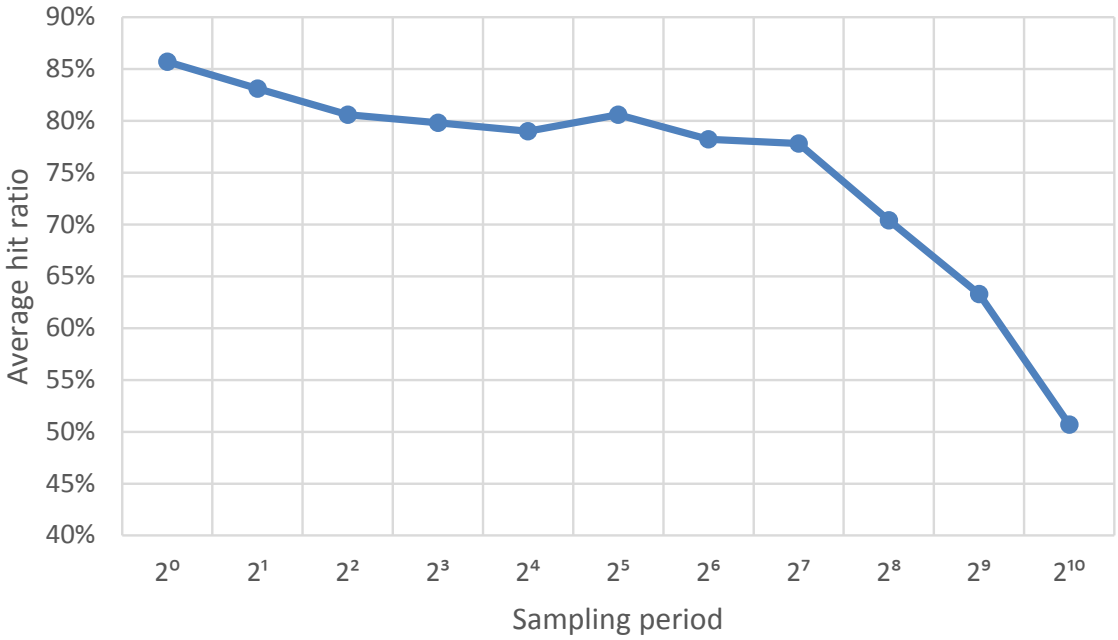

Figure 3.9: Impact of sampling period on cache hit ratio

3.2.5 Read Sampler Sampling Period

To reduce the storage requirement for RRD prediction table, a reasonable sampling period needs to be adopted. We conducted an experiment to explore the impact of sampling frequency on the hit ratio of L2 cache as shown in Fig. 3.9. We found when the sampling period increases from 2^3 to 2^7, the average hit ratio barely changes across PARSEC benchmarks. However, from 2^7 to 2^{10}, it started to fluctuate and decreased significantly at 2^{10}. This is because a large sampling period is not able to capture the locality in the access stream. Since a small sampling period introduces intensive PC comparison and table updates, which impairs predictor performance in turn, we chose a sampling period of 2^7 in our final evaluation. The size of sampling buffer can be determined via the equation:

$$Sampling\_FIFO\_size = \frac{Max\_RRD}{Sampling\_Period} \tag{3.4}$$

The maximum RRDs we observed in memory traces are generally under 2^{10}, and the sampling period is set as 2^7. Thus we selected 2^3 as the FIFO size.

The size of the sampling buffer is 2^3=8 elements deep in our design. That means the buffer can store 8 PCs at most. When a write request is sampled at the sampling stage, its PC is stored as a stall (i.e., a void state) in the buffer. This is because we concentrate on read operations in the access stream. If the write-associated PCs are stored, the predictor will produce general reuse distance of the cache block. On the other hand, if the write-associated PCs are skipped, then the predicted RRD will diverge from accurate behavior.

Table 3.1: Baseline Configuration

CPU	4 cores, 3.3 GHz, Fetch/ Exec/ Commit width 4
L1	private, 32 KB, I/D separate, 8-way, 64 B, SRAM, WB
L2	private, 4 MB, unifed, 8-way, 64 B, STT-RAM, WB
Main Memory	8 GB, 1 channel, 4 ranks/ channel, 8 bank/ rank

3.3 Evaluation

3.3.1 Experiment Setup

The evaluation was conducted by using the cycle-accurate simulator MARSSx86 [53]. We modified its cache controller module to realize the proposed function. The simulator mimics the computer architecture as shown in Table 3.1.

Eleven different benchmarks from PARSEC 2.1 suite [10] were used for the experiment, executing 500 million instructions starting at the Region Of Interest (ROI) after warming up the cache with 5 million instructions. We counted read and write accesses on odd (HBL) and even (SBL) cache lines using simsmall and simlarge input sets respectively. Table 3.2 shows the similar cache line access distribution for these two input sets. Thus, similar experimental conclusions are obtained by choosing either of them. The simsmall input sets are selected for all benchmarks.

We adopted the serial MLC STT-RAM cell design from [77] and scaled it under 32nm technology node [2]. The small MTJ pillar is configured as 32nm$\times$64nm elliptical shape. We used the NVSim and CACTI [27][67] to obtain the key design parameters as shown in Table 3.3. MLC STT-RAM model is integrated into NVSim by modifying configurations, such as set/ reset currents, nMOS transistor sizes, etc. The energy contributions from peripheral circuits are also included.

Table 3.2: Cache Line Access Distribution

	simsmall				simlarge			
	read_even	**read_odd**	**write_even**	**write_odd**	**read_even**	**read_odd**	**write_even**	**write_odd**
fluidanimate	24%	21%	33%	22%	24%	22%	33%	21%
swaptions	24%	22%	28%	26%	24%	22%	28%	26%
facesim	22%	20%	38%	20%	22%	19%	38%	21%
dedup	23%	19%	37%	21%	21%	18%	39%	22%
vips	20%	17%	35%	28%	21%	19%	38%	22%

Table 3.3: Comparsion of 4 MB SLC STT-RAM, MLC STT-RAM and SRAM [27][67]

	SLC STT-RAM	MLC STT-RAM	SRAM
Array Area (mm^2)	1.86	1.01	7.28
Read Latency ($Cycles$)	9.08	S: 6.73 H: 9.80	7.43
Write Latency ($Cycles$)	25.58	S: 25.31 H: 56.50	5.78
Read Energy (nJ)	0.216	S: 0.22 H: 0.43	0.161
Write Energy (nJ)	0.839	S: 0.843 H: 2.502	0.156
Leakage (mW)	18.39	7.02	295.58

3.3.2 Write Disturbance Restore Overhead Reduction

The normalized numbers of restoration reduced by ARS-WD are shown in Fig. 3.10. The baseline MLC has 100% restoration. When applying ARS-WD, 54.6% of the total restore operations are avoided on average; in some benchmarks like streamcluster, where there are a large portion of distant read intervals, ARS-WD can save up to 62.7% of restoration.

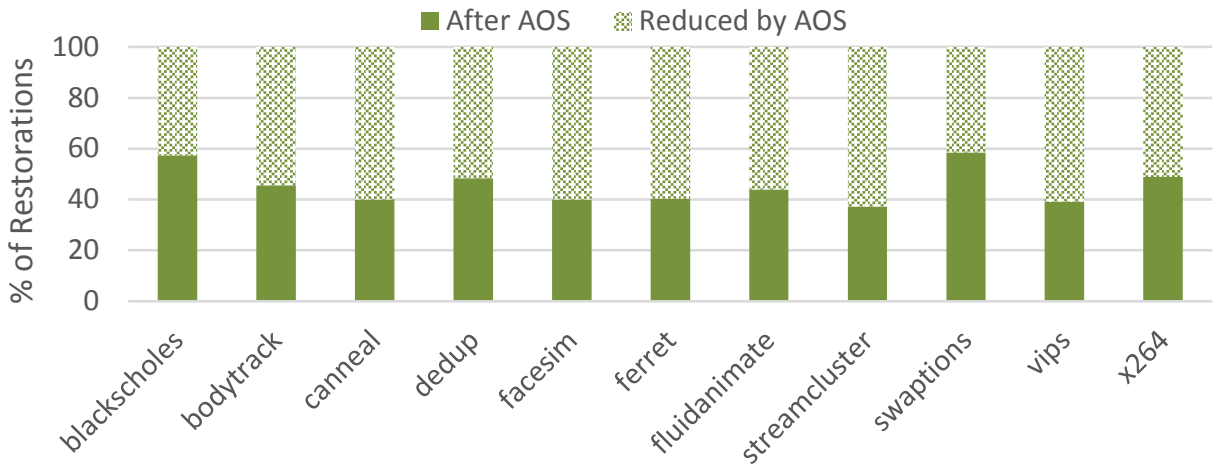

Figure 3.10: Restoration Reduction with ARS-WD

3.3.3 Energy Area Latency (EAT) Product Comparison

There are a few other technologies that can be employed as L2, for instance, SLC STT-RAM and conventional SRAM. SLC almost doubles the cell area of MLC with the same capacity, whereas MLC suffers from asymmetric read and write performance and needs restoration. On the other hand, SRAM consumes significantly more leakage power and size in contrast to MTJ-based technologies. We use Energy Area Latency (EAT) product [7] as metrics to not only find the energy efficiency ARS-WD can help to improve, but also to evaluate the preferred L2 candidate.

In the following evaluations, the baseline is the MLC with conventional immediately restore scheme. No SBL restoration can be deducted in the baseline.

3.3.3.1 Energy Comparison

Fig. 3.11 compares the dynamic energy consumption of four L2 candidates. SRAM consumes the least dynamic power, as switching transistors from "on" to "off" and "off" to "on" are symmetrical. Writes are not as power hungry as they are in STT-RAM. With ARS-WD, 10.2% of dynamic

energy in MLC is reduced on average.

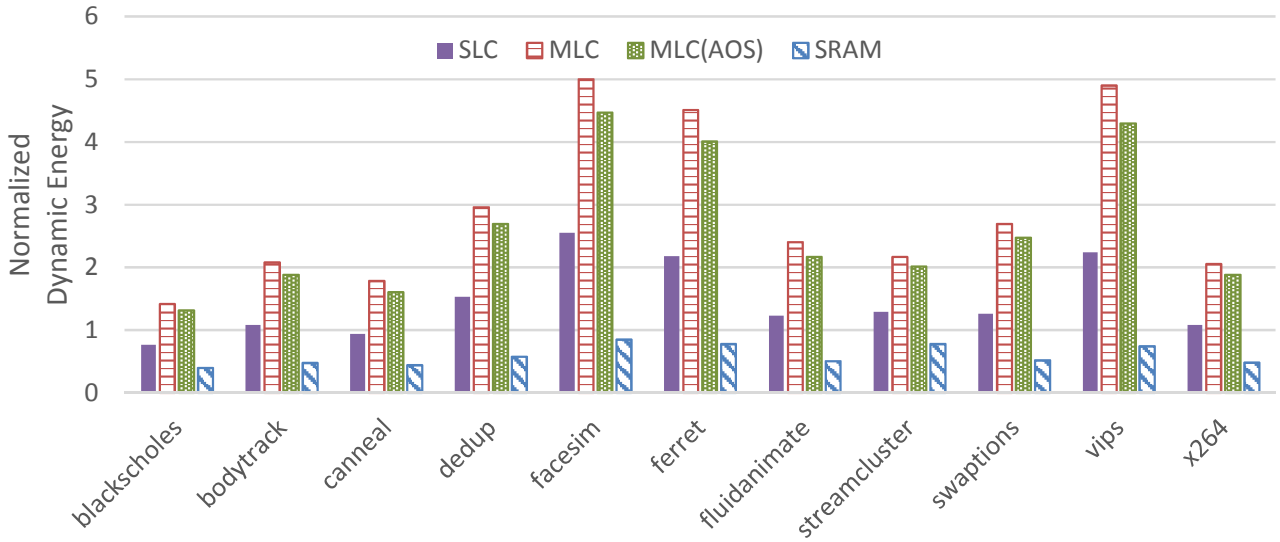

Figure 3.11: Dynamic energy comparison among different L2 candidates

To calculated the leakage energy of four candidates, we used the execution time of each benchmark and the unit leakage power. The overall energy consumption equals the sum of dynamic and leakage energy as shown in Fig. 3.12. SRAM consumes considerably more leakage energy than other candidates, making the overall energy consumption enormous. ARS-WD can mitigate 10.8% of the total energy for MLC on average. In write intensive benchmark like vips, ARS-WD can save up to 13.0% energy.

3.3.3.2 Latency Comparison

We used the read and write numbers and unit latency to calculate cache access latency. ARS-WD reduces unnecessary restorations and decreases MLC latency by up to 14.85% and 11.72% on average as shown in Fig. 3.13. SRAM is the fastest among four candidates due to its symmetrical rapid access speed. Note here the overhead incurred by the peripheral circuits are also included.

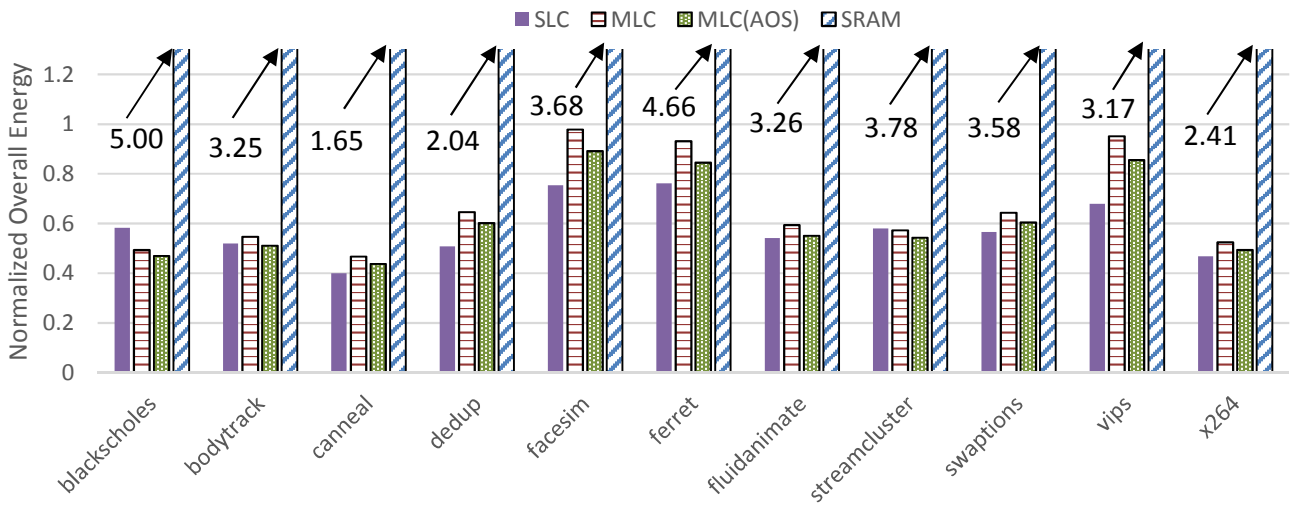

Figure 3.12: Overall energy consumption among different L2 candidates

3.3.3.3 EAT Product

Fig. 3.14 shows that MLC with ARS-WD has the preferable EAT in most cases. Compared to SLC, which tends to be considered as the preferred L2 candidate by intuition, MLC with ARS-WD decreases EAT by 4.6% on average. In the read intensive workload like streamcluster, EAT is reduced by 20.3%. This is because SLC almost doubles the size of MLC, and besides, the read latency of SLC is very close to that of the hard bit in MLC. ARS-WD also reduces the baseline MLC EAT by 10.1%.

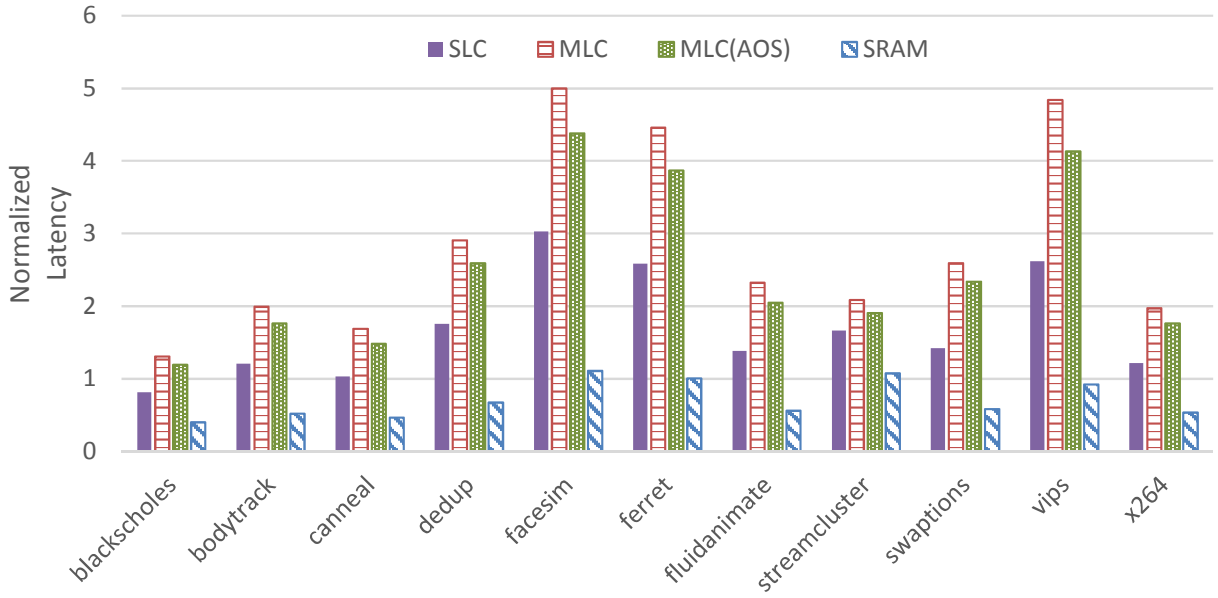

Figure 3.13: Cache latency comparison among different L2 candidates

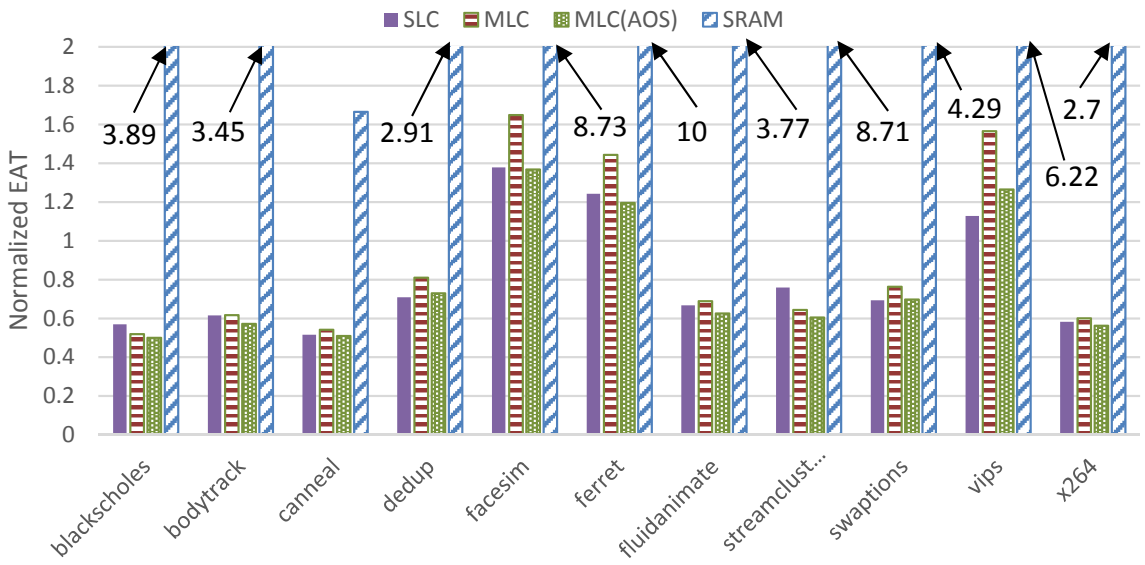

Figure 3.14: EAT comparison among different L2 candidates

CHAPTER 4: ADAPTIVE RESTORE SCHEMES FOR READ DISTURBANCE

As the result of MTJ feature size scaling, the soft bit can be expected to become disturbed by the read sensing current, thus requiring an immediate restore operation to ensure the data reliability. In this chapter, we design and analyze a novel Adaptive Restore Scheme for Read Disturbance (ARS-RD). ARS-RD aggregates the potential writes and restore the soft bit line at the time of its eviction from higher level cache. ARS-RD is also based on a lightweight forecasting approach for the future read behavior of the cache block. Our experimental results show substantial reduction in soft bit line restore operations, delivering 17.9% decrease in overall energy consumption and 9.4% increase in IPC, while incurring negligible capacity overhead. Moreover, ARS promotes advantages of MLC to provide a preferable L2 design alternative in terms of energy, area and latency product compared to SLC STT-RAM alternatives.

4.1 Read Disturbance

In the section, we first present the bit error rate caused by read disturbance, then the significant energy overhead of IRS in RD (IRS-RD) is demonstrated. Lastly, we propose ARS-RD which is based on RRD to alleviate the restoration overhead.

4.1.1 Motivation

In contrast to the switching current, the sensing current does not continue to scale down with the feature size [66]. Fig. 4.1 shows the comparison between read and write currents at different technology nodes. For large feature sizes, such as 130nm, the read current is much smaller than

the write current. Therefore, reading data out of the MTJ will not accidentally flip the stored state. Nevertheless, for the smaller process technologies, it is arduous to shrink the read current amplitude, since conventional STT-RAM sense amplifiers are unable to guarantee the data sensing accuracy using below 20μA current[80]. Thus, the read current remains relatively constant in the deep sub-micrometer regimes, and the margin between write and read current values diminishes significantly. For example, at 32nm node, the sensing current approaches the switching current so closely that some reads may accidentally write their being-read cells[66]. Equation 3 below is adopted to model the read disturbance rate:

$$P = 1 - \exp\left\{-\frac{t}{\tau}\exp\left[-\Delta_0\left(1 - \frac{I}{I_{c0}}\right)\right]\right\} \tag{4.1}$$

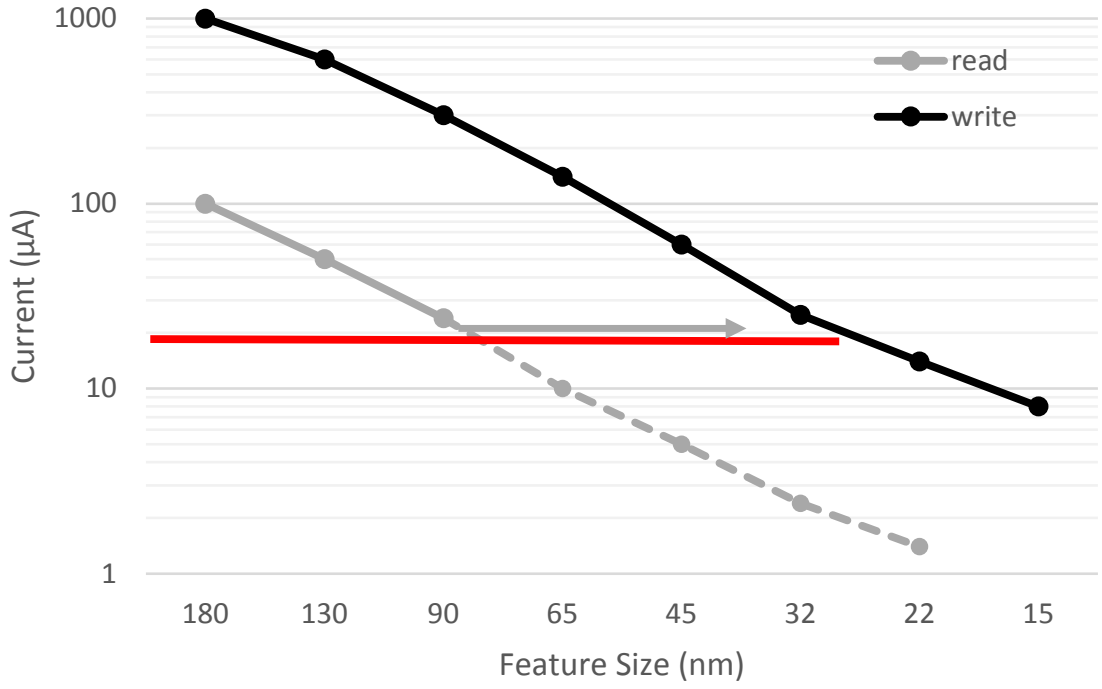

Figure 4.1: Read and Write Current Scaling [66]

where P donates the read disturbance rate; I is the read current value; t is the read pulse duration; τ indicates inverse of the attempt frequency; Δ_0 donates the magnetic memorizing energy without any impact from current or magnetic field. I_0 indicts the critical switching current at 0K. Previous works [71][35] have utilized this model to estimate the Raw Bit Error Rate (RBER). At 15nm node and beyond, the BER with strong ECC, such as BCH, is still larger than the acceptable error rate for on-chip caches (1E-3)[75]. Therefore, similar to the solution for WD, data is immediately restored back after every read operation, which is referred as Immediate Restore Scheme for Read Disturbance (IRS-RD) as shown in Fig. 4.2.

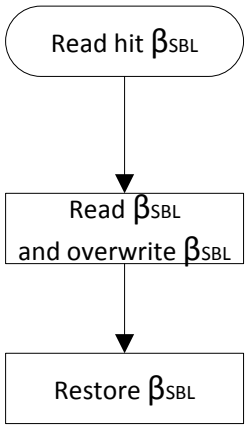

Figure 4.2: Immediate Restore Scheme for Read Disturbance

4.1.2 Read Access Pattern Analysis

The cache block in LLC can be categorized as reused and non-reused. A reused block is the one accessed for more than once from upper level, while the non-reused one does not experience multiple accesses during its residency in LLC. As shown in [40], the majority of read accessed cache blocks in LLC are non-reused. This portion of cache blocks takes a relatively small fraction of program execution time as they are read only once. In contrast, reused cache block with large amount of

read access rate experience portion of read operations. For example, these cache blocks consume more than 16% of program execution time while take only less than 2% block count in LLC. These reused cache block experience either read only access or mixed access. The read intensive with more 64 access time account for around 8% of total read access. This group is referred as Extensive Read Reused Access (ERRA) which remains stable during program execution. These observation motivates Khoshavi et al. to propose a cache migration policy and architecture that favor ERRA blocks, which will reduce read response time and increase system IPC. Recent work [62][64] also prioritize the performance critical read reused block in order to enhance overall performance. The author propose to arrange ERRA blocks to the high retention STT-RAM due to two reasons. The first is experience read only access after the initial write. The second reason is read introduces much less overhead than write operation, it will only be accessed by read before its eviction.

After each LLC access, no matter it is a hit or miss, the system will find a victim block in higher level for replacement of the LLC block. Because of the non-inclusive model as we described before, the clean victim will be forced out from higher level silently during the replacement, while the dirty block is inserted into LLC. However, the replica may not exist in LLC still for because of the non-inclusive property. This requires a victim block evict from LLC or dirty victim inserted from higher level to LLC. Khoshavi et al. investigates the access pattern of block of which the residency in higher level can decrease the miss rate. Nevertheless, these blocks evict from higher level before major contribution to the read hit rate owing to the weak temporal locality, inefficient replacement policy or insufficient capacity to keep the highly referred datasets [58]. Thus, whenever there is a read miss on those blocks, the system has to promote the referred block from lower level to higher level which causes performance degradation and energy consumption. However, these blocks may not be accessed again during its stay in higher level. About 20% of the evicted clean block from higher level will be re-used more than 16 times later. In other words, when the evicted clean victim block is required, the processor stalls for these cache block migration. This

incurs significant energy dissipation and delay for the multi-level cache system.

4.1.3 Energy Cost of Handling Read Disturbances

Assuming N_R read requests occur in total, then the energy consumed by the IRS for RD strategy (E_{IRS_RD}) is given by Equation 4. Whereas reading a cache line will also destroy the SBL stored value, cache block is copied to the write buffer. The address is decoded and restored upon every read, which incurs energy of E_{PC} and E_{WSBL}, respectively. Note here only the soft bit cells are susceptible to read disturbance, as the surface area of the hard bit cell is at least twice as large as the area of the soft bit cell [77], i.e. 1.4 times in terms of feature size. The sensing current is large enough to corrupt the SBL data regardless of soft or hard bit access.

$$E_{IRS\_RD} = (E_{PC} + E_{WSBL}) \cdot N_R \tag{4.2}$$

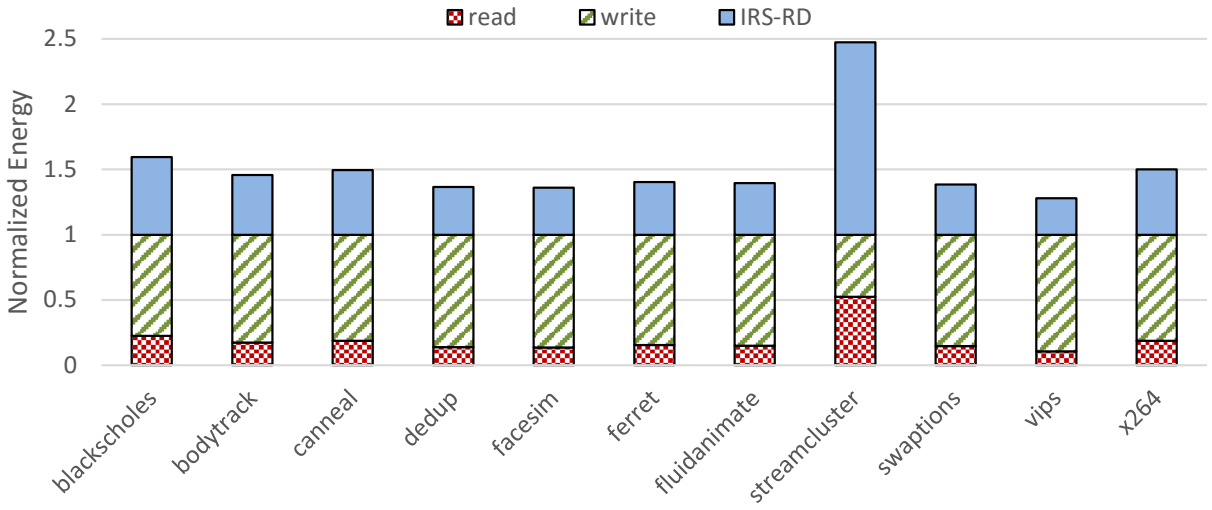

Figure 4.3: Dynamic Energy Breakdown of MLC STT-RAM with RD

The dynamic energy consumption breakdown of read, write and IRS for RD (IRS-RD) with PAR-

SEC benchmarks is depicted in Fig. 4.3. IRS consumes 52% additional dynamic energy on average and up to 147% in read-intensive workloads.

4.1.4 Adaptive Restore Scheme for Read Disturbance

However, in RD, it is unnecessary to immediately restore a block which will be modified and written back. Aiming to merge the restorations of disturbed blocks with dirty block write-backs, Adaptive Restore Scheme for Read Disturbance (ARS-RD) is presented in this section.

ARS-RD is a two-fold scheme as shown in Fig. 4.4. Ideally, a cache line requires at most one restore operation after being read from L2. To this end, ARS-RD first fetches the block to L1 without immediately correcting it, then the victim block is selectively restored to L2 based on the its modification status, origination and associated RRD.

- Since only the SBL will be disturbed in the read operation, an 'S' bit is attached to each L1 cache line to differentiate the block read from an SBL. As shown in Fig .3.2 & Fig. 4.4(a), when β_{SBL} is read, its replica in L1 should carry the modification status to ensure the data integrity. To this end, we attach one more flag bit 'C' to every cache line in L1 to indicate whether a block is unmodified or not when loaded from L2 [71]. Note here the dirty bit in L1 is not inherited directly from that in L2, because that can increase the number write-back to L2. In particular, when a modified block is fetched to L1 for read only, due to the inheritance of dirty bit, it will be written back to L2 anyway at the time of eviction. Lastly, L2 invalidates the disturbed block after every read without writing back it immediately, and then the block is copied to L1.

  On the other hand, when α_{HBL} is read, the 'S' bit is set as '0'. Next, the block is brought to L1 directly. IRS or even ARS-WD can be applied to address the disturbance issue of its

corresponding block β_{SBL}.

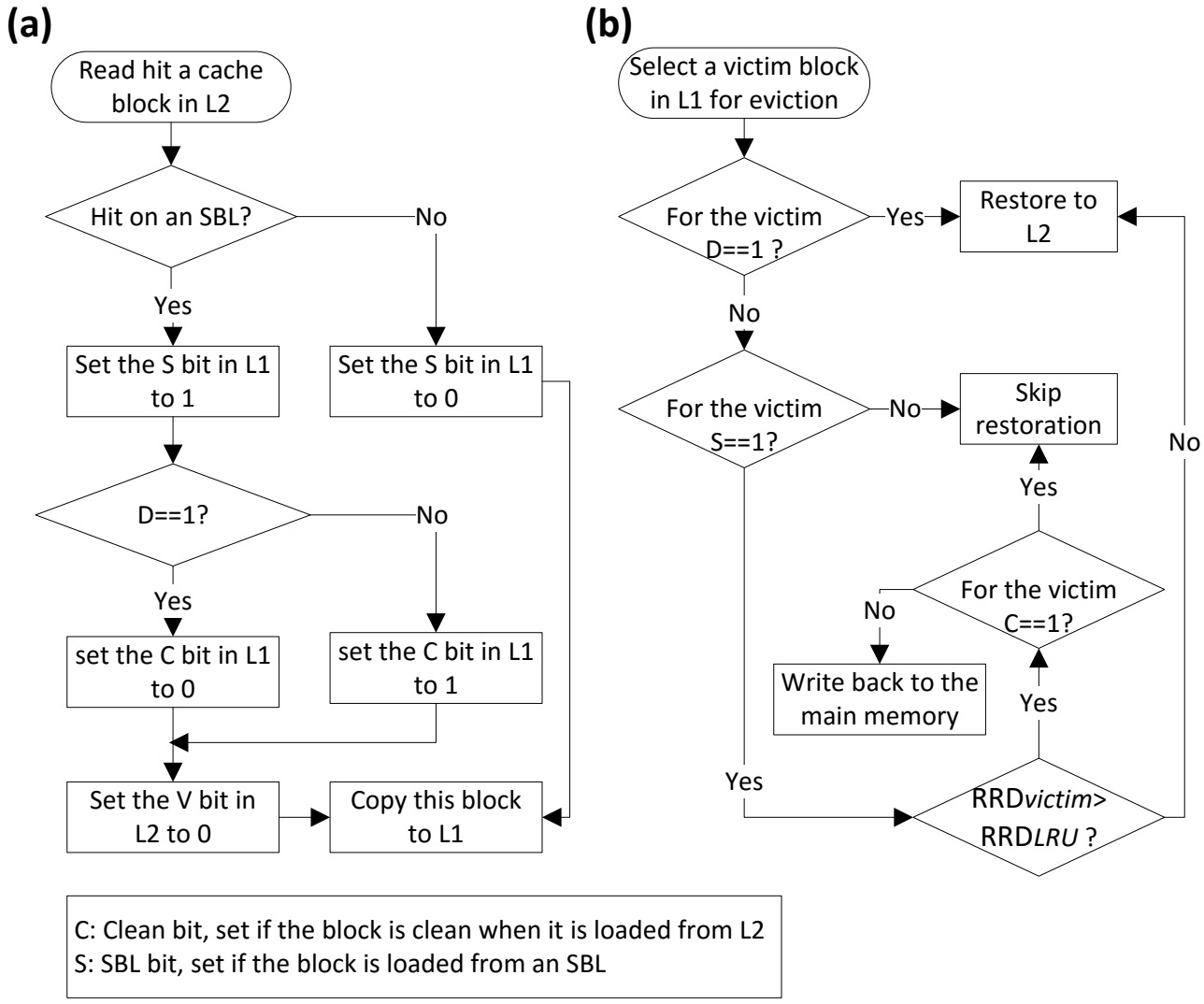

Figure 4.4: Adaptive Restore Scheme for Read Disturbance (ARS-RD)

- Upon the eviction of a block in L1 as described in Fig. 4.4(b), its dirty bit is first checked for a write-back decision. If the block has been modified, it will be restored to L2. While if the block is clean, it cannot be simply discarded as specified in the conventional write back policy. This is because an unmodified block in L1 is originated from either the cache fill or

the read fetch on L2 (Fig. 2.6). For the first origination, the clean block can be abandoned, while for the second one, the block with 'S' bit set is fetched with the disturbance and invalidation of its copy in L2. Discarding it will lead to a data integrity issue.

- Due to the foregoing reasons, L1 checks the 'S' bit of the clean victim block. If it is not set, that implies this block was loaded either from HBL read or cache fill. The victim can be dropped directly as there is no block disturbance involved. Otherwise, it is necessary to consider whether or not β_{SBL} still exists in L2's replacement policy stack if it was not invalidated initially.

- Next, we compare the RRD of the victim (RRD_{victim}) with that of the first block in L2's replacement policy stack (RRD_{LRU}). It has been proven that read misses are more performance critical than write misses, as read misses will delay the processor [62]. Therefore, we protect the block which is likely to serve read requests (RRD_{victim} is smaller than RRD_{LRU}) by restoring it to L2. Otherwise, L1 tests the 'C' bit for the next decision.

- If the victim is never written('D' is '0' and 'C' is '1'), read reuse distance is approximate to reuse distance [39]. Since the victim block is less probable to be accessed than the first one in replacement policy stack at this stage, it will be discarded directly. While if the 'C' bit is not set, the evicted block is written back to the main memory to ensure the data integrity.

We utilize C for reducing number of L2 write-back. Here, the flag C is used to indicate whether a block is modified (C=1) or not (C=0). Thus, C reflects if this block is clean or not in L2 when it is loaded. There is another dirty bit in L1 to indicate whether the block is modified afterwards. A similar approach have been utilized in [71], whereby the status of cache block is tracked while it is transferred among levels of cache.

ARS-RD differs from the previous Direct Restore (DR) scheme [71] in two major aspects. First,

after a block in L2 is read and disturbed, it is invalidated directly in ARS-RD instead of remaining in L2. This effectively improves the cache utilization. Second, after a victim is selected for eviction in L1, the scheme in [71] requires to check if the victim exists in L2. However, considering the large capacity of L2, this consumes significant resources. Based on RRD, ARS-RD trades cache capacity for performance. Additionally, unlike DR which can only solve RD issue, ARS-RD is coupled with ARS-WD to provide a holistic solution to both RD and WD issues in MLC. Due to these trade-offs and in pursuit of the stated goals of this paper, the evaluation of DR has not been emphasized in the experimental results.

4.2 Read Sampler and Prediction Table Case Study

It has been observed that a specific instruction often executes a highly unique task (e.g., memory read access) and rarely changes this behavior. As instructions are uniquely associated with their Program Counters (PCs), which describes the instruction address in memory, PCs provides a very effective way of recording program context and predicting program behavior. An alternate explanation for the validity of PC-based cache access behavior predictor can been found in related works such as [43][44][57], some of which utilize workloads similar to those used in this study.

4.2.1 Read Sampler

The Read Sampler samples the memory access stream periodically and store the associated PC into a buffer. If it is a write, put a stall in buffer. On each read access, it searches in the Sampling buffer for a matching PC. If match, an (instruction, read reuse distance) pair is produced.

Fig. 4.5 shows how the read sampler calculates read reuse distance. Assume we have a memory access stream as shown in the figure, RdA comes first and end with RdC. The read sampler sampled

the data every two accesses, therefore PCs associated with RdA, RdC are sampled into FIFO buffer because they are read operations. In Fig. 4.5(1), PC1 of RdA is first sampled to the buffer, and the tail pointer moves to the next position. The the next memory access is WrtB as shown in Fig. 4.5(2), since it is a write operation, the sampler will not match the PC of WrtB in the buffer, so there is no action needed here. In Fig. 4.5(3), read sampler tries to match PC2 of RdC among the existing PCs in the buffer. Tail pointer moves to the next position as read sampler cannot match PC2. In Fig. 4.5(4), RdC is sampled since it is a read operation, PC2 of RdC will be stored into the buffer. The next PC for matching process is the one associated with RdD as shown in Fig. 4.5(5), since there is no PC in the buffer for memory address D, the tail point moves to the next position. Next, the read sampler samples WrtB. However, since this is a write operation, the PC of WrtB will not be stored into the sampling buffer. As shown in Fig. 4.5(6), a stall will be stored in the buffer as a placeholder. The next memory access is a write operation WrtC and it will not trigger PC match. In the end as shown in Fig. 4.5(7), read sampler samples RdC and it matches an existing PC in the buffer. The read reuse distance is calculated by the following equation:

$$Read\_Reuse\_Distance = FIFO\_Position * Sampling\_Period \tag{4.3}$$

In this example, the relative FIFO position is 2 and sampling period is 2, so the estimated read reuse distance equals to 4.

4.2.2 Read Reuse Prediction Table

Once the Read Sampler generates a pair of RRD and PC, it goes into RRD prediction table as shown in Fig. 4.6 and Fig. 4.7. The prediction table is constructed in cache style, which is addressed by the PC (instruction) and holds the read reuse distance of the PC as well as a saturating Confidence

Counter (CC). This predication table is responsible for two major functions.

The first one is to update RRD based PC as shown in Fig. 4.6. First the system find the entry for the given PC. If the incoming new RRD and the stored RRD are identical, the CC increases by one; while if they are different, the CC decreases by one; else when the CC equals to zero, the system will simply update the stored RRD in the entry with new RRD.

The second function is to lookup the corresponding RRD based on PC as shown in Fig. 4.7. When an instruction causes a LLC access and we want to predict its read reuse behavior. The system first tries to find the entry for the given PC. If the entry was found and the confidence counter is above the threshold, the system returns the saved read reuse distance. If not, the system temporarily cannot make accurate predication.

4.3 Apply ARS To Different Inclusion Models

As we discussed in Section 2.4, there are inclusive, non-inclusive and exclusive models used in multilevel cache hierarchy. This paper adopts non-inclusive model, while non-inclusive cache has some limitations. e.g., cache coherence issue. Thus, in order to prove more general utility of the proposed ARS method, we will discuss how to apply the idea to the inclusive and exclusive architectures.

In [5], Ahn et al. proposed a novel write energy reduction technique at architecture level for STT-RAM based LLC. This technique does not need extra SRAM cache space or device level enhancement. The key observation made here is a large amount of writes to LLC, which are caused by write-back from higher level or block fills, can simply bypass the the LLC without introducing any extra cache misses. As described in [45], more than 80% of cache blocks are not reused before they are replaced in a 2MB LLC. And more than 25% of these blocks are never accessed again.

Thus, it is better to bypass these block instead of inserting them into LLC. For the rest block more than 55% of cache block that will be accessed, the Belady's OPT [8] replacement policy can be applied here. Belady's OPT with bypass scheme first selects the block with minimal reuse distance to fill up the cache, then bypass the others to avoid unnecessary writes. Also, bypassing the incoming new blocks when their reuse distance is equal to or larger than the victim block that is chosen by the baseline. More importantly for emerging write expensive NVM, bypass can reduce the energy consumption caused by write back from higher level and replacement. In [45], the author proposed Optimal Bypass Monitor (OBM). OBM learns the optimal bypass behavior to facilitate future decision making. A small Replacement History Table (RHT) is introduced to track new victim block. Upon every cache access, the pair of new block and replacement candidate will be compared with data in RHT to determine the optimal bypass behavior. A Bypass Decision Counter Table (BDCT) which is indexed by program counter, is to the dominated behavior and decide if bypass should be applied. OBM is compatible with the current LLC design and replacement policies. OBM introduces very small hardware overhead is capable of multi-thread, prefetch.

The proposed Dead Write Prediction Assisted STT-RAM Architecture (DASCA) bypasses a write to LLC when it is predicted not to incur any extra cache misses. The author designed and implemented a write predictor to forecast the type of write and whether it is a dead write. The bypass scheme advocated in the paper is bidirectional, that means, either from main memory to higher level cache or from high level cache to main memory. The scheme is ideal for non-inclusive cache hierarchies without the need for much modification. However, when the inclusive model is enforced in the multilevel cache hierarchy, it is hopeless to directly adopt DASCA bypass scheme as the upward one unavoidably violates the inclusive nature. One of the solutions is to arrange the dead write blocks into replacement stack rather than bypass them [45].

Although this method can deliver same amount of miss rate reduction, it does not favor write energy reduction because the block is inserted and will lead to write operations anyway. The

author introduces an extra cache block state 'void' in this scenario. DASCA inserts cache block into replacement slot without actually writing the data content and the block status is marked as void. When a void block is accessed, it is handled similar to a cache miss and will trigger cache fills. The cache coherence state bit is updated just as the block is valid. The introducing of this void bit reduce the STT-RAM overall write energy while keep the inclusive property.

We leveraged the non-inclusion property to enable ARS in this work. However, when an inclusive cache hierarchy is enforced, simply applying ARS violates the nature of inclusion. One way to address this issue is to introduce an additional 'void' state [5]. In particular, ARS can be applied to inclusive cache by keeping the disturbed block as 'void' in the LRU chain. Since accesses to void blocks are handled as cache misses, it is possible to develop a replacement policy which considers the void state and minimizes the total miss penalty based on RRD. On the other hand, ARS can be readily adopted to the exclusive cache hierarchy. Note here in ARS-WD, the step of checking the duplicated block in higher level cache can be skipped because of the exclusion property. In ARS-RD, it is unnecessary to restore the disturbed block after read operations as a block can only exist at one level due to space constrain.

4.4 Evaluation

In this section, we first introduce our simulation platform setup. Then we evaluate MLC STT-RAM with ARS in terms of energy, latency, EAT product and IPC. For the experiment platform setup, we adopt the same experimental setting as described in shown in section 3.3.1 (shown in Fig. 4.8).

We have conducted the experiment of counting read and write accesses on odd and even cache lines again using simlarge input set, and observed similar cache access distribution as follow in Fig. 4.9. Since our EAT evaluation is based on this distribution, we expect the same conclusion

if simlarge is used. Due to page limitation, we have only selected five of the benchmarks to show that both simsmall and simlarge input sets exhibit similar cache access behavior. In addition, we found simsmall is widely used in the research published in the leading conferences and journals. For example [37][3][4][59][28].

4.4.1 Read Disturbance Restore Overhead Reduction

The percentages of dynamic energy consumed by IRS and ARS are shown in Fig. 4.10. As presented in Section 3.3 and Section 4.3, IRS-WD and IRS-RD dissipate 23.1% and 51.9% of dynamic energy respectively. After applying ARS-WD, 54.6% of the WD restore operations are avoided on average, decreasing the WD restore energy overhead to 10.6%; in some benchmarks like streamcluster, where there are a large portion of distant read intervals, ARS-WD can save up to 62.7% of restoration. On the other hand, ARS-RD reduces 36.9% of RD restorations and consumes only 35.1% dynamic energy on average. In the write intensive workloads, such as vips, a significant amount of restorations (59.0%) are merged with the dirty block write-back.

4.4.2 IPC Comparison Between ARS-RD and DR

Two major aspects are presented in Section 4.1.4 that how the proposed ARS-RD differs from the previous Direct Restore (DR) scheme. We compare the IPC between ARS-RD and DR as shown in Fig. 4.12 below. Our scheme outperform DR in terms of IPC by 5.4% on average, but requires more space (2% more space) to store RRD information. Additionally, DR is not able to alleviate the write disturbance issue in MLC STT-RAM, while our work aims to provide a holistic solution to read and write disturbance issues in MLC. Due to these trade-offs and in pursuit of the stated goals of this paper, the evaluation of DR has not been emphasized.

4.4.3 Energy Area Latency (EAT) Product Comparison

There are a few other technologies that can be employed as L2, for instance, SLC STT-RAM and conventional SRAM. SLC almost doubles the cell area of MLC with the same capacity, whereas MLC suffers from asymmetric read and write performance and needs restoration. On the other hand, SRAM consumes significantly more leakage power and size in contrast to MTJ-based technologies. We use Energy Area Latency (EAT) product as metrics to not only find the energy efficiency ARS can help to improve, but also to evaluate the preferred L2 candidates.

In the following evaluations, the baseline is the MLC with conventional IRS by default. No SBL restoration can be deducted in the baseline. Similar to the soft bit in MLC, SLC will have RD issue after scale down to 32nm and beyond. We consider four L2 candidates, e.g. SLC with ARS-RD, MLC with IRS, MLC with ARS, SRAM in our comparison.

4.4.3.1 Energy Comparison

Static and dynamic energy consumption can be compared as follows. Considering SRAM, energy consumed due to its large cell structure is exacerbated by subthreshold and gate leakage. Similarly, to implement the cache with same capacity, SLC doubles the number of transistors and thus has higher leakage power than MLC. Further overviews of dynamic and static energy consumption in various device technologies have been presented in [14].

Fig. 4.11 compares the dynamic energy consumption of four L2 candidates. SRAM consumes the least dynamic power, as the switching process of CMOS transistors in SRAM is symmetrical and much easier than changing the resistance states of MTJ in STT-RAM. Thus, writes in SRAM are not as power hungry as they are in STT-RAM. With ARS, 17.3% of dynamic energy in MLC is reduced on average.

To calculated the static energy of four candidates, we used the execution time of each benchmark and the unit leakage power. Considering SRAM, energy consumed due to its large cell structure is exacerbated by subthreshold and gate leakage. Similarly, to implement the cache with same capacity, SLC doubles the number of transistors and thus has higher leakage power than MLC.

Fig. 4.14 compares the average dynamic and static energy consumption of four L2 candidates over PARSEC benchmarks. The leakage energy of SRAM exceeds its dynamic energy consumption by more than an order of magnitude, while MLC and SLC dissipate significantly more dynamic energy than their static component.

The overall energy consumption equals the sum of dynamic and leakage energy as shown in Fig. 4.13. SRAM consumes considerably more leakage energy than other candidates, making the overall energy consumption enormous. ARS can mitigate 17.9% of the total energy for MLC on average. In write intensive benchmarks like vips, ARS can save up to 21.4% energy. Furthermore, MLC with ARS consumes less power than SLC with ARS-RD in read intensive benchmarks, e.g., blackscholes, streamcluster.

4.4.3.2 Latency Comparison

We used the read and write latency and number to calculate the overall cache access latency. ARS reduces unnecessary restorations and decreases MLC access latency by up to 23.8% and 18.8% on average as shown in Fig. 4.15. SRAM is the fastest among four candidates due to its symmetrical rapid access speed. Note here the overhead incurred by the peripheral circuits are also included. The latency gap between SLC and IRS-based MLC expands in the write-intensive workloads due to the high write latency associated with HBLs. For instance, the access delay of SLC is 43.4% less than that of IRS-based MLC in vips. However, the gap diminishes quickly in read-intensive workloads, i.e. 25.4% in streamcluster.

4.4.3.3 EAT Product

For a given capacity, MLC-based STT-MRAM cache can decrease the area. EAT product is defined as the numerical product of Energy, Area, and Latency. SLC doubles the area compared to MLC regardless of workloads due to the storage of two bits within two MTJs of a memory cell. Meanwhile, for read intensive workloads, the read latency of SLC is very close to that of the hard bit in MLC. Thus, EAT is dominated by area in this case which results in significant reduction in EAT product for MLC design.

Fig. 4.16 shows that MLC with ARS has the preferable EAT in most cases. Compared to SLC, which tends to be considered as the preferred L2 candidate by intuition, MLC with ARS decreases EAT by 2% on average. In the read intensive workload like streamcluster, EAT is reduced by 35.5%. SLC doubles the area compared to MLC regardless of workloads due to the storage of two bits within two MTJs of a memory cell. Meanwhile, for read intensive workloads, the read latency of SLC is very close to that of the hard bit in MLC. Thus, EAT is dominated by area in this case which results in significant reduction in EAT product for MLC. ARS also substantially reduces the baseline MLC EAT by 32.9%.

4.4.4 IPC Comparison

In contrast to the access latency, IPC indicates the overall system performance. Fig. 4.17 compares the IPC of four L2 cache configurations with similar array area: 2MB SLC(IRS), 2MB SLC(ARS-RD), 4MB MLC(IRS), 4MB(ARS). Here the 4MB MLC with IRS is taken as the baseline. The IPC of SLC(IRS) is decreased by 6.5% on average, while for some benchmarks that do not significantly benefit from the larger cache capacity, such as fluidanimate, IPC is only reduced by 1.7% due to the longer access latency of HBLs. ARS-RD slightly enhances the IPC of SLC(IRS) by

5.3%. Although ARS-RD discards a large amount of unnecessary RD restorations, dynamically computing RRD and comparing it take extra clock cycles. However, MLC with ARS substantially improves the IPC of SLC(IRS) by 17.6%. This is not only due to the larger cache capacity, but also because of the reduction of WD and RD restoration. Compared to the baseline, ARS also improves the IPC by 9.4%.

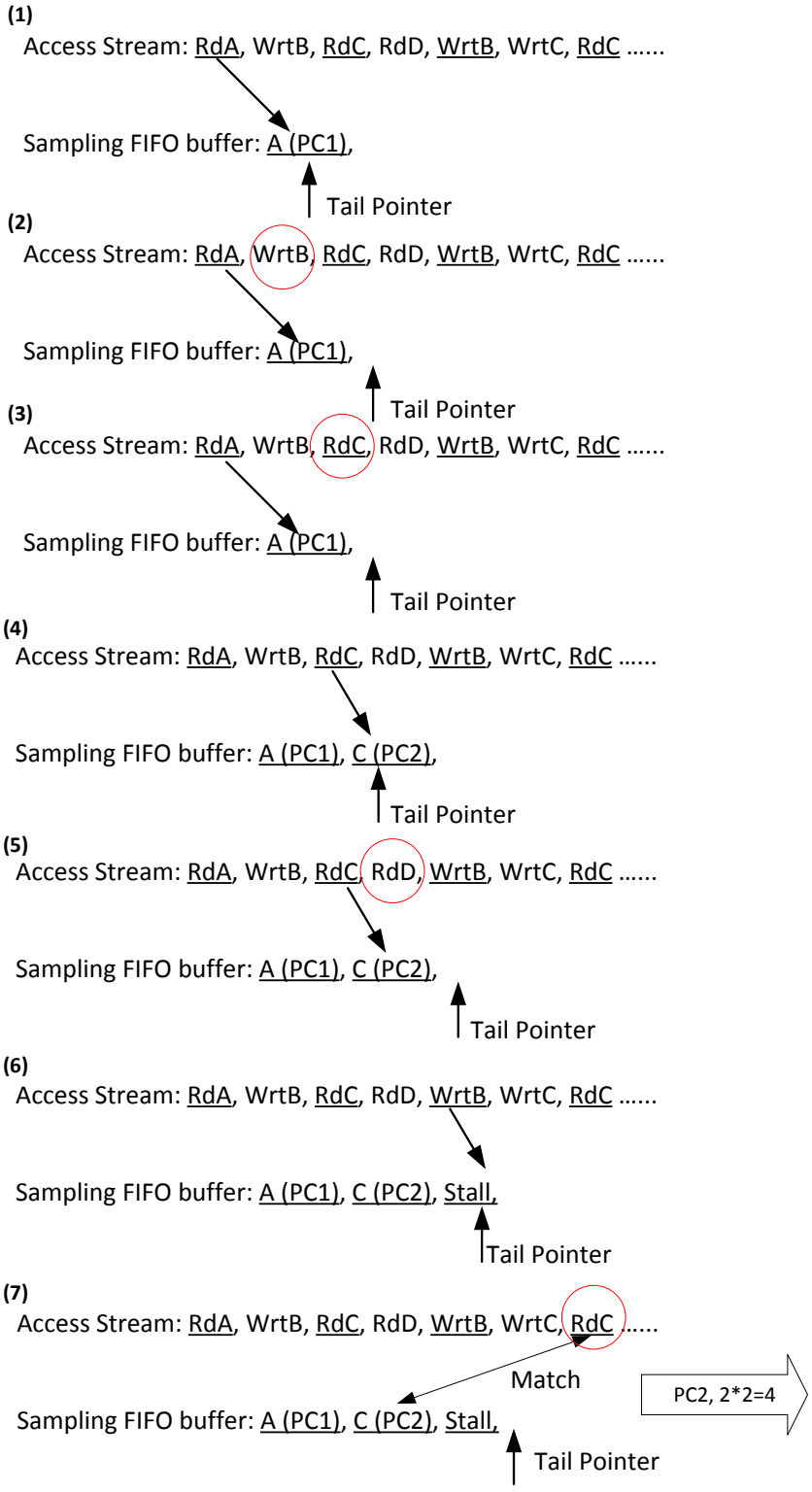

Figure 4.5: Read Sampler

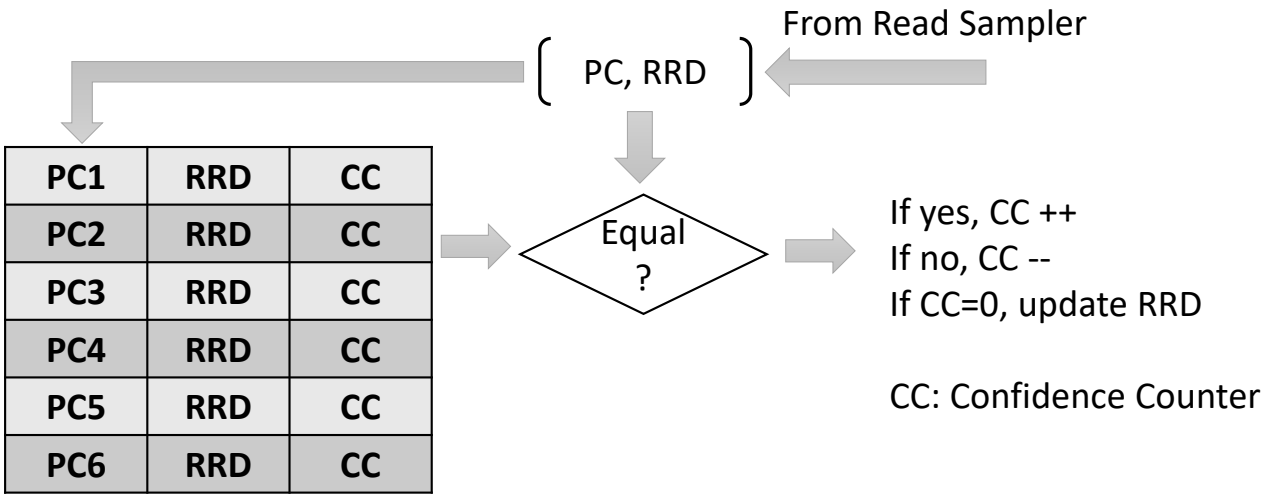

Figure 4.6: Read Reuse Distance Prediction Table Update

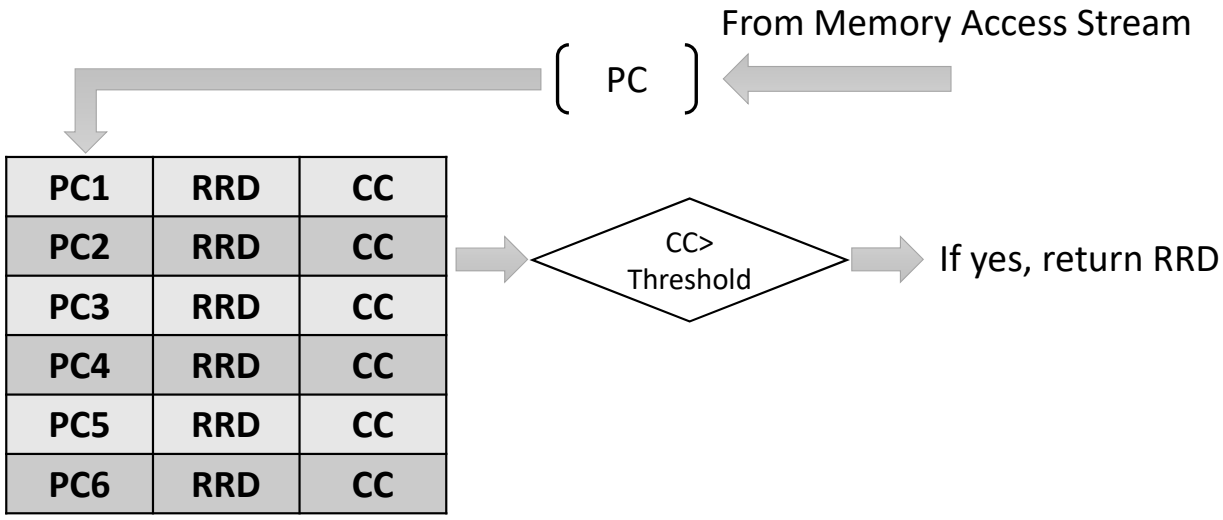

Figure 4.7: Read Reuse Distance Prediction Table Lookup

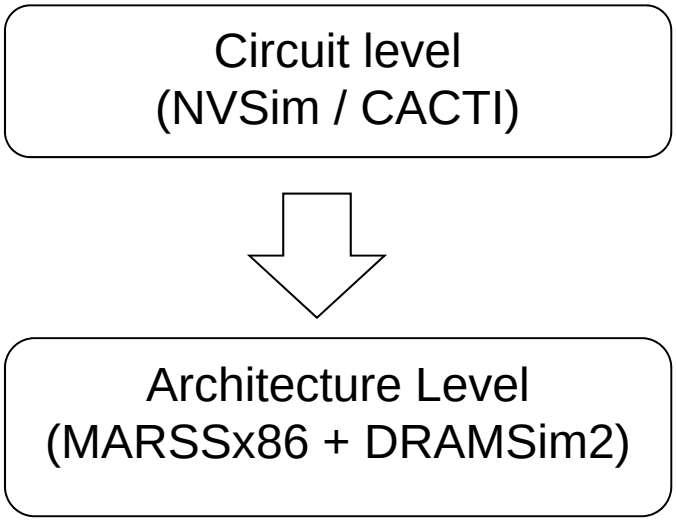

Figure 4.8: Experiment Settings

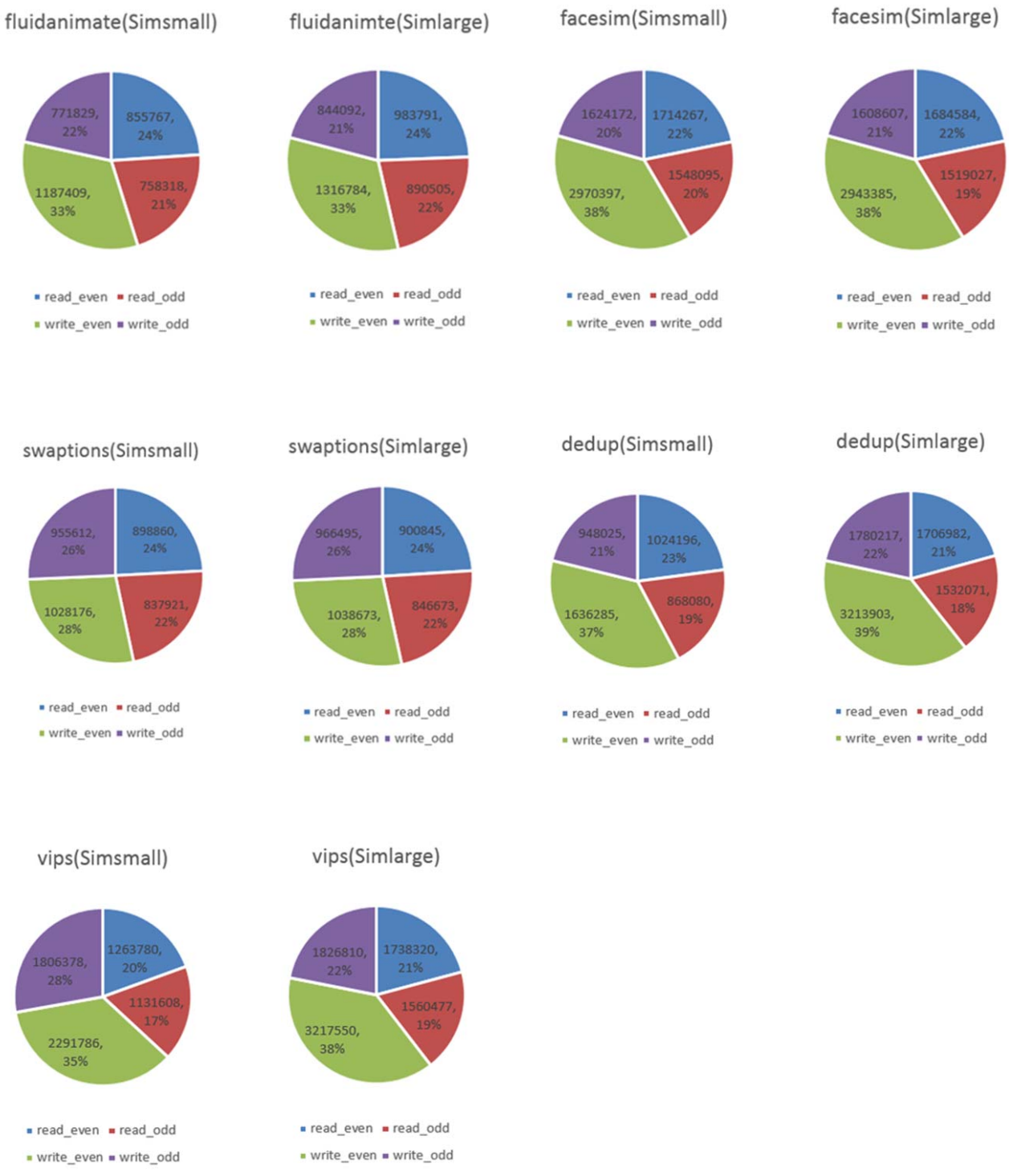

Figure 4.9: Simsmall and Simlarge comparison

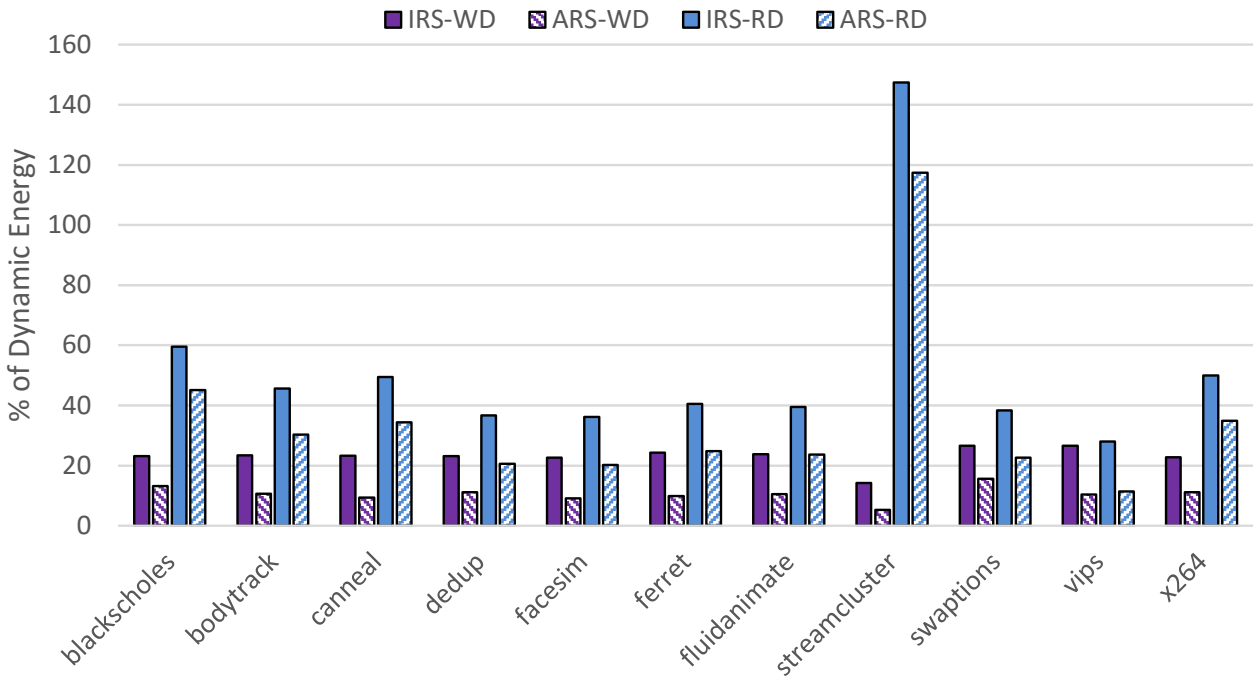

Figure 4.10: Restoration Overhead

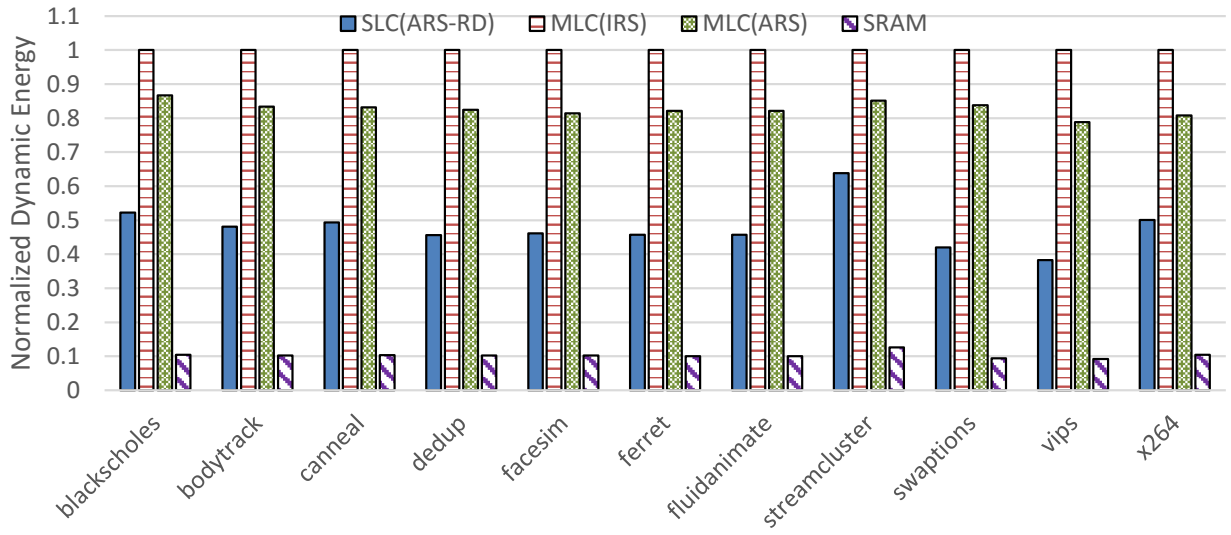

Figure 4.11: Dynamic energy comparison among different L2 candidates

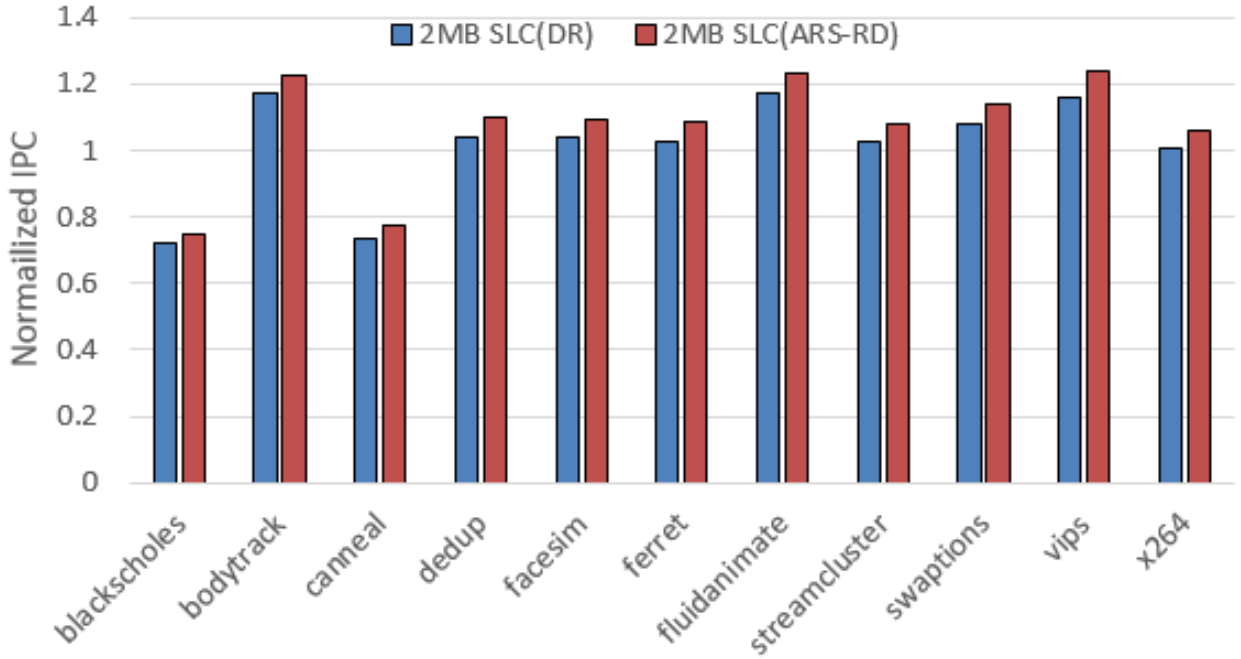

Figure 4.12: IPC Comparison Between ARS-RD and DR

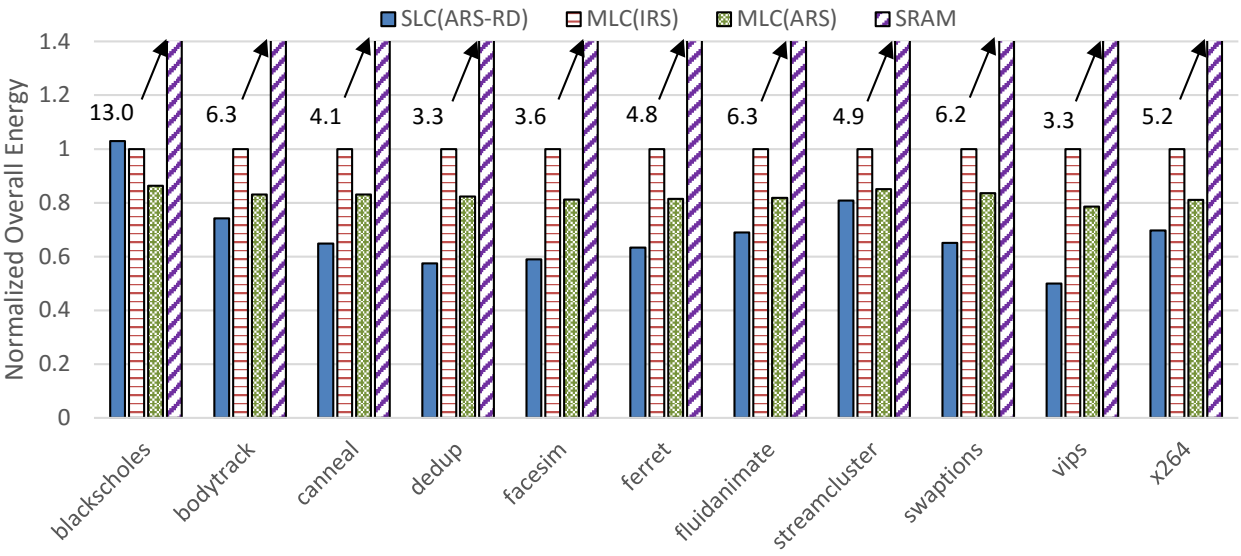

Figure 4.13: Overall energy consumption among different L2 candidates

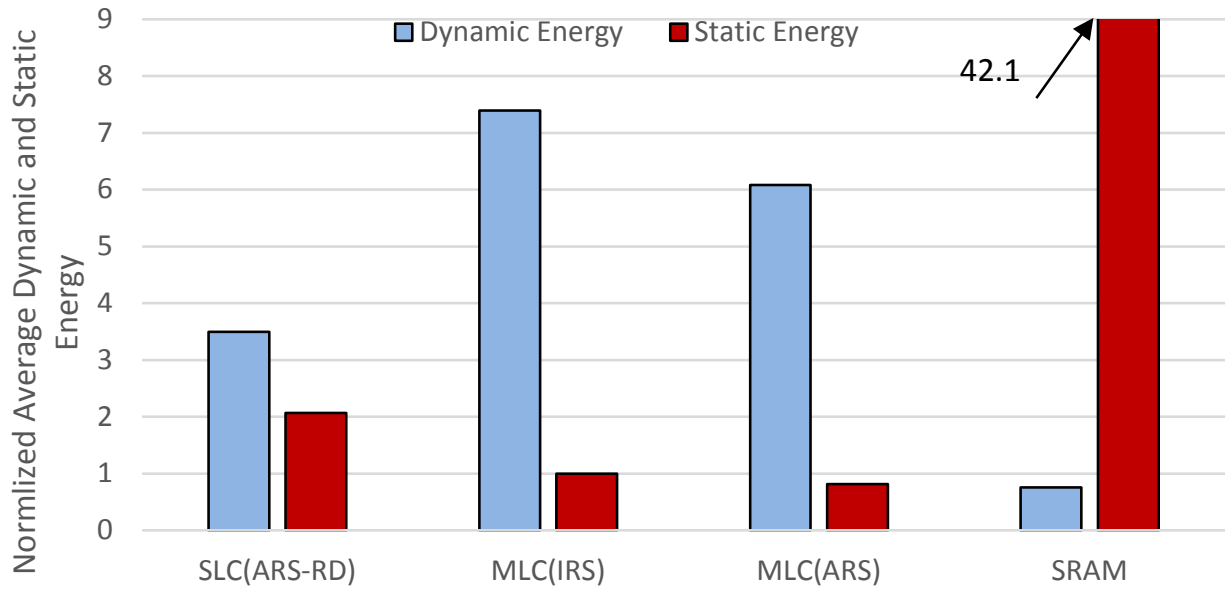

Figure 4.14: Average dynamic and static energy consumption comparison among different L2 candidates

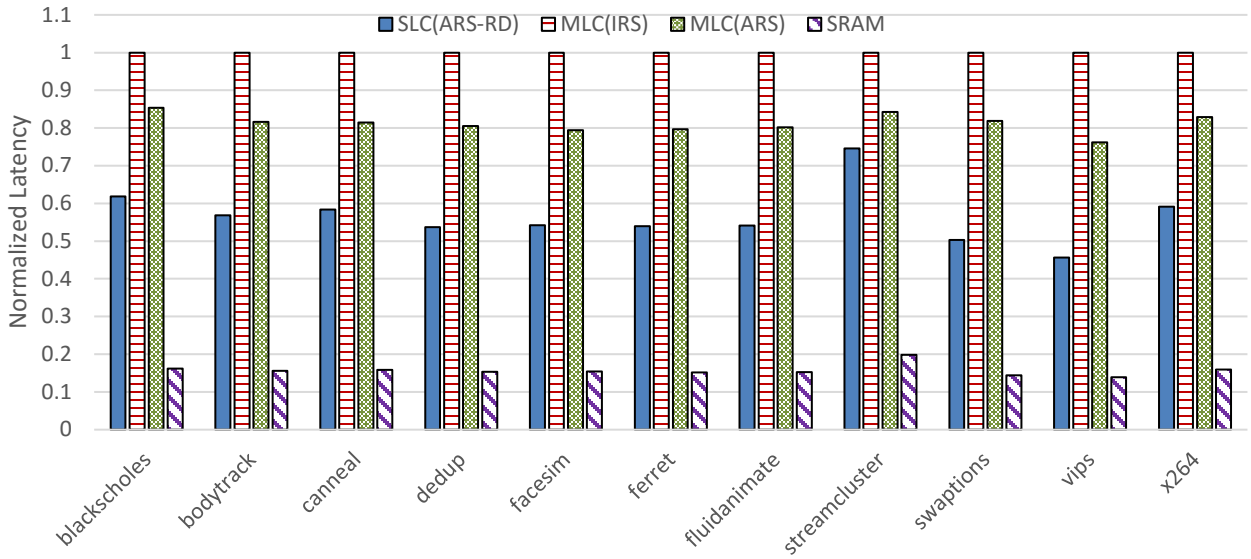

Figure 4.15: Cache latency comparison among different L2 candidates

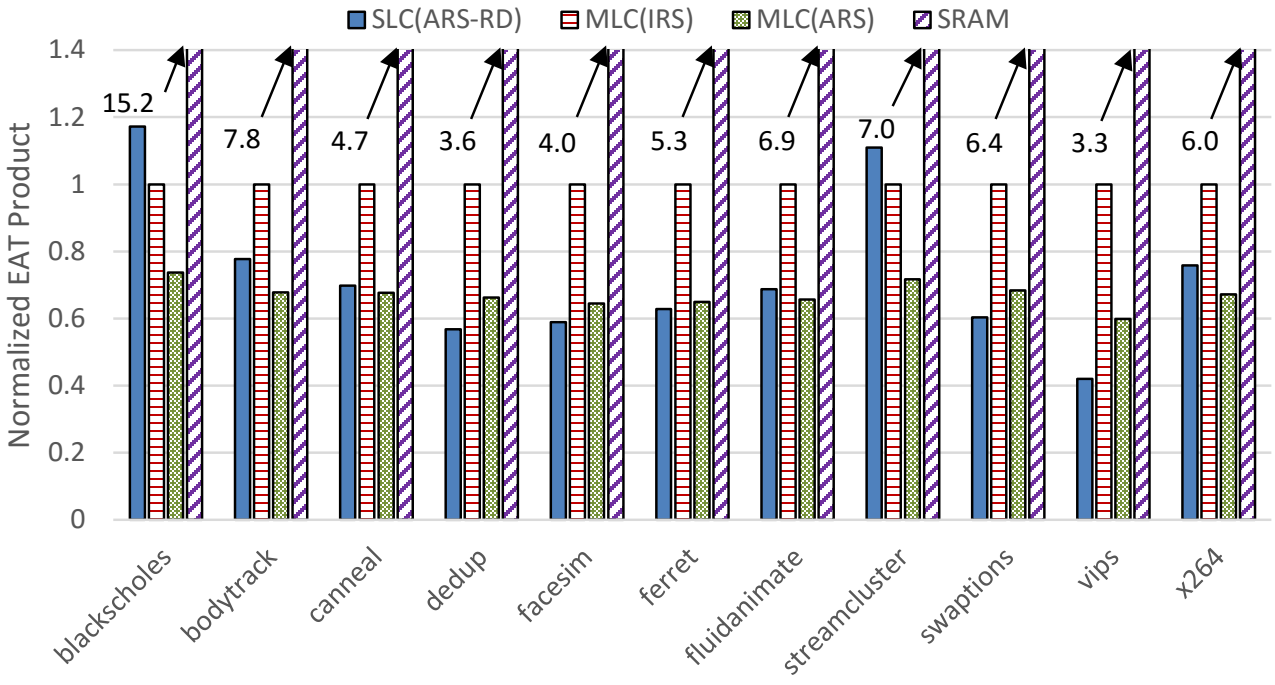

Figure 4.16: EAT comparison among different L2 candidates

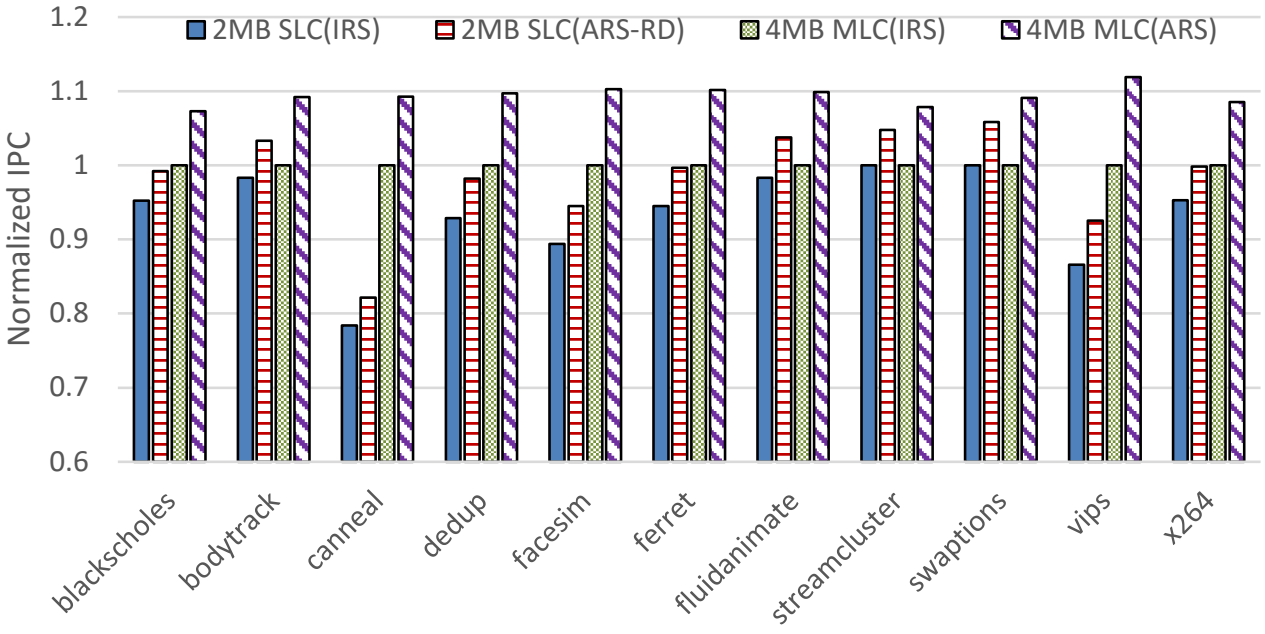

Figure 4.17: IPC comparison among L2 configurations with similar array area

CHAPTER 5: DOUBLE-S HYBRID MAPPING STRATEGY

Aiming to minimize the energy-hungry write requests to HBL and maximize the dynamic range in the advantageous SBL, an hybrid mapping strategy for MLC STT-RAM cache (DoubleS) is advocated in this paper. DoubleS couples the contemporary Cell-Split-Mapping with the novel Word-Split-Mapping (WSM). Sparse cache block detector and read depth based data allocation/ migration are proposed to release the full potential of DoubleS.

5.1 Employing MLC STT-RAM in Persistent Main Memory System

5.1.1 Emerging Persistent Main Memory

Due to scalability and energy consumption issues of traditional DRAM and SRAM memory technologies, researchers start to explore new materials as memory which is more scalable or do not rely on electronic charge effect [19]. NAND flash SSD has already been used in the persistent memory system. Other promising candidates include Phase Change Memory (PCM), Spin-Transfer Torque Random Access Memory (STT-RAM), and Resistive Random Access Memory (ReRAM). These emerging memory technologies use resistance value instead of electrical status to store data. Some attractive features emerging memories include high scalability for integration, fast read access, non-volatility. Since we have introduced STT-RAM in details in Section 2, we will focus on NAND flash, PCM and ReRAM in this section.

As shown in Fig. 5.1, in a flash cell [12][13], on top of it is the Control Gate (CG) which is connected to word line. Under the CG is the Floating Gate (FG). FG is sandwiched between two insulators, which are typically inter-poly oxide and tunnel oxide. The electrons charged on the FG will stay even when the power is off which make the transistor with FG non-volatile. The

source and drain of FG transistor are shared with it neighboring transistors. Compared to a normal transistor, the FG transistor adds a floating gate under CG which is used to store electrons.

Similar to a normal transistor, the voltage on the CG generates a conductive channel between the source and the drain. The minimal voltage that enable the channel is referred as threshold voltage. This threshold voltage can be tuned by injecting charges to FG. During read disturbance or program operations, charges are introduced to FG and cause Fowler-Nordheim (FN) tunneling effect. This effect generates an electric tunnel between FG and substrate. The charge on FG creates a electric field and offset the field generated by CG. The strength of this field is proportional to the voltage on CG, also the charges on FG.

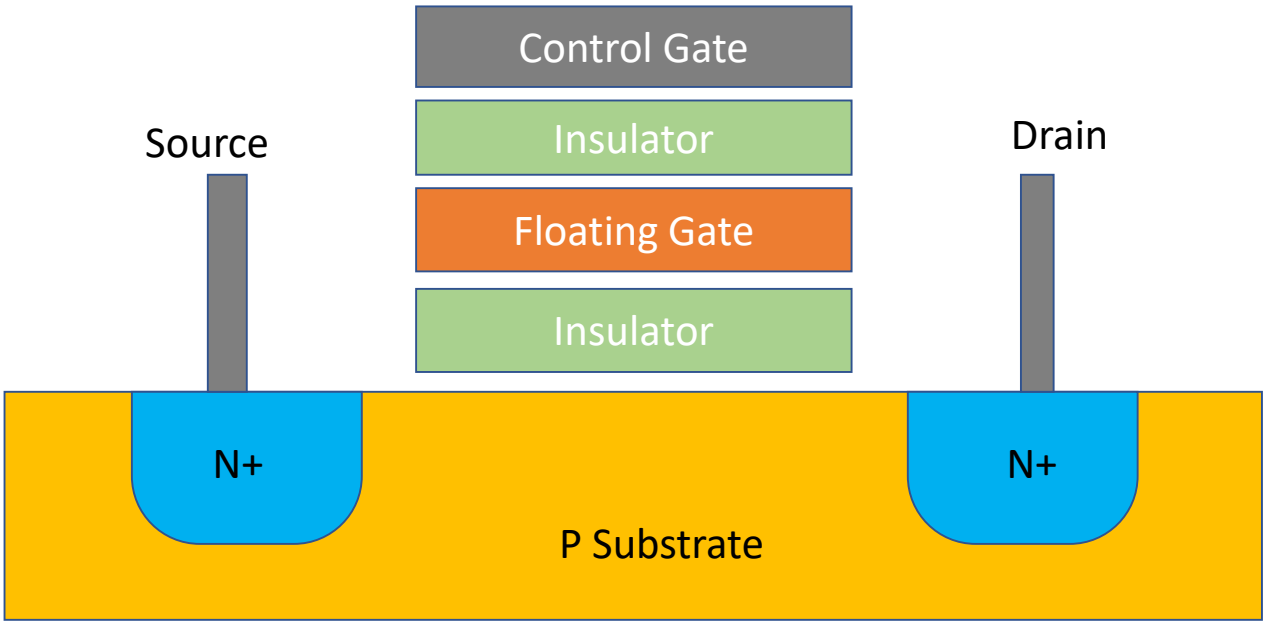

Figure 5.1: Cross Section of A Flash Cell

As suggested by its name, phase-change material changes its phase to control the resistance value. There are two phases in PCM, referred as amorphous phase and crystalline phase. The first one has high resistance and low reflectance, while the second one has low resistance and high reflectance.

The material can switch between these two phases under certain condition. The resistivity difference of these two phases can be used to represent logic 1 and 0.

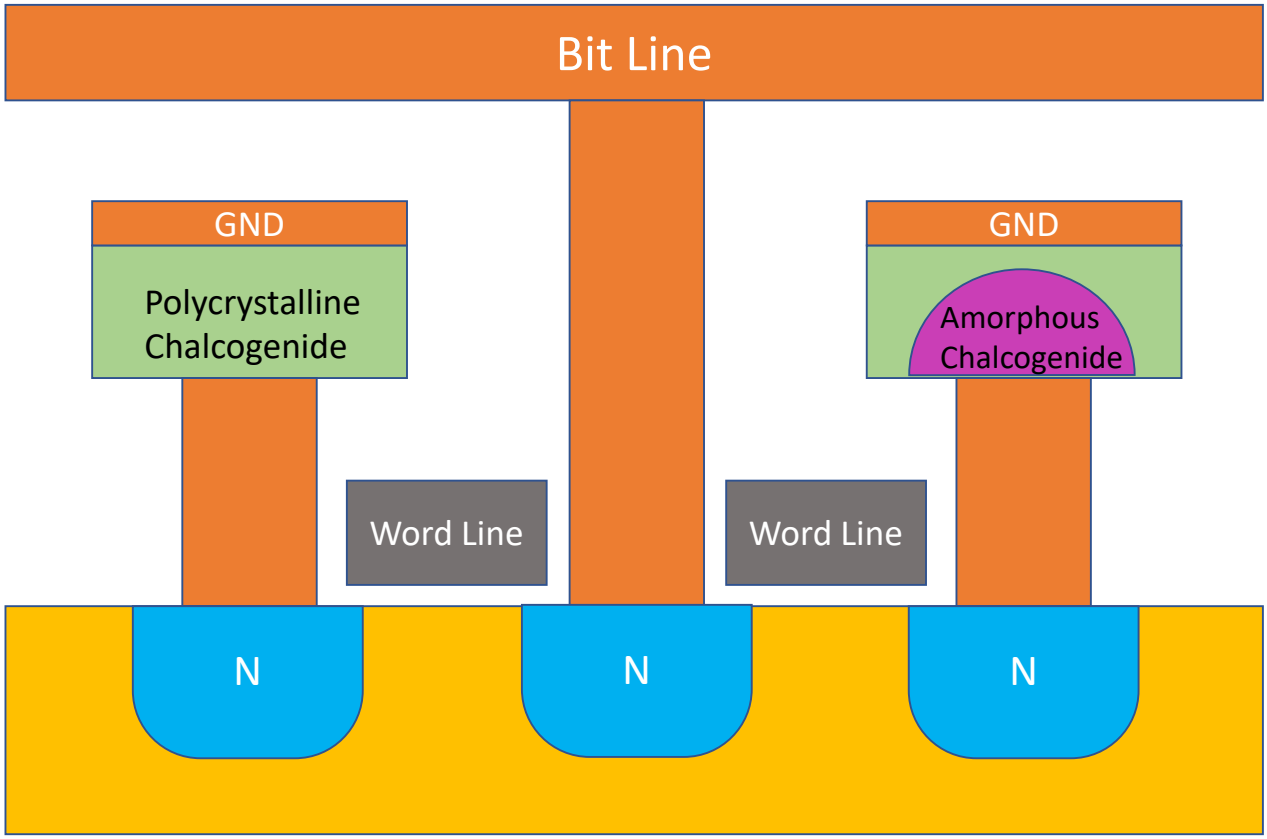

Figure 5.2: Cross Section of Two PCM Cells

As shown in Fig. 5.2, one PCM cell is in low resistance crystalline state, the other in high resistance amorphous state. Phase change material such as chalcogenide is integrated between electrodes. A heating component which is typically a resistor, is placed on the bottom electrode and connect to the chalcogenide layer. The programming operation of a PCM includes two steps, namely SET and RESET.

In the SET step, a relatively long and moderate current is injected into the heater. The temperature of chalcogenide layer increases above the crystallization point and then it is crystallized. In this

scenario, PCM is in low resistance state. While during the RESET step, a short and high current is induced and increases the temperature of the programmed volume beyond the melting point. As a result, the material polycrystalline order vanishes and the programmable volume turn into amorphous again. In this scenario, PCM is in high resistance state. The reflectivity changes with resistance, thus, the the programmable volume appears as a mushroom structure in the figure.

Whereas PCM and STT-RAM are already in production, ReRAM is under study by researchers. ReRAM usually leverages metal-oxide materials to store data. A layer of metal-oxide material is put between two electrodes. In the process of RESET, the material turns from low resistance state into high resistance state. The oxygen ions combines with vacancies and travels from electrode-oxide interface.

On the left side of conductive filament region, a high resistance per unit length is formed. While a low resistance per unit length is formed on the right side. For the SET process, the material switches from high resistance state to low resistance state. By injecting a current or voltage with certain pulse widths or magnitudes, the material can be programmed to different resistance states, in other words, different logic values.

5.1.2 Challenges Of Employing NVM As Main Memory

Currently, there are still several serious challenges in using NVMs as the main memory.

The first one challenge is writing to NVM cells is much more expensive than reading in terms of performance and energy consumption. NVMs, including STT-RAM, PCM and NAND flash, usually have limited write endurance. To reduce the write and prolong and the lifetime of NVM, many techniques have been proposed such data comparison write, flip-n-write and dead write bypassing. For MLC STT-RAM specifically, although it has higher cell density than SLC STT-

RAM, it suffers from read disturbance and write disturbance issues. Upon every read to the MLC, the value in it can be disturbed. Similarly, upon every write to the hard bit line of MLC, the corresponding soft bit line will be disturbed.

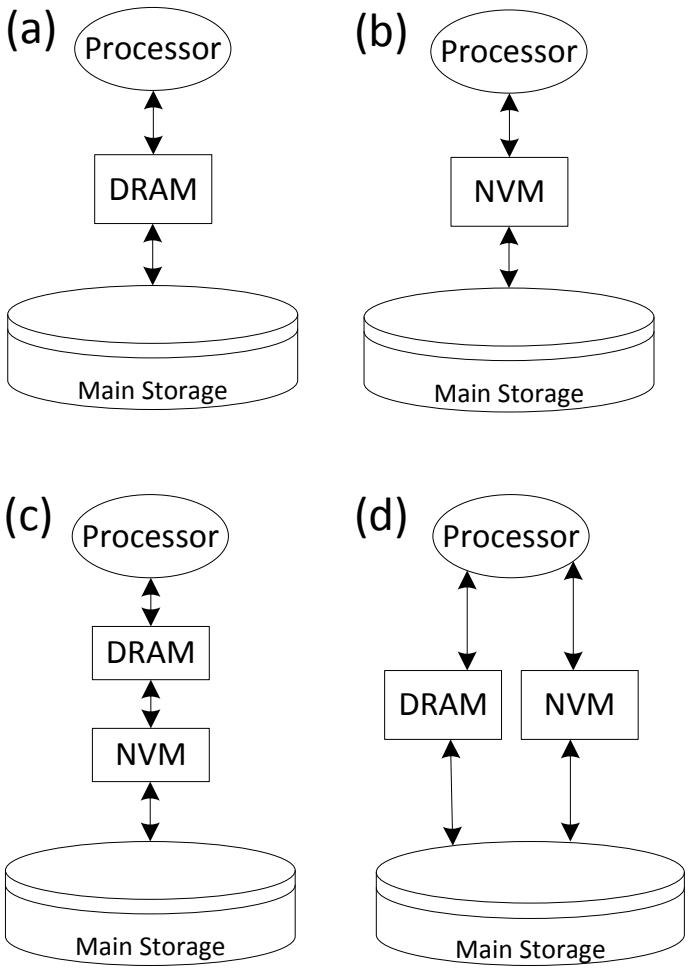

Figure 5.3: (a) Traditional DRAM Main Memory (b) DRAM Main Memory Replaced By NVM (c) NVM Deployed As Storage Cache (d) NVMDeployed As Memory Extension

Another challenge is the architecture choice for the persistent memory system. NVM can be adopted as a direct replacement of the DRAM main memory as shown in Fig. 5.3(a) & Fig. 5.3(b).

Whereas this architecture can be adopted for the low-frequency and high energy-efficiency pro-

cessor, the high performance computer system may not favor this architecture. This is because the overall system performance can be degraded by long write latency of NVMs. For those applications that do not require large memory capacities and so that the memory miss penalty is not very significant, the computer equipped with traditional DRAM is predicted to outperform the one with NVMs because of the long write latency.

Another deployment architecture of NVM is a storage cache between main memory and storage layer as shown in Fig. 5.3(c). However, I/O software path, such as file system can largely devalue the advantages of NVMs. The last user case of NVM is deploying it the memory extension as shown in Fig. 5.3(d). Memory extension, or sometimes called Storage Class Memory (SCM), is a secondary level memory which cooperate with the DRAM based main memory. The operating system will choose which memory pages are held in DRAM main memory and which pages are moved to the memory extension.

5.2 Intra-block Data Access Feature

Whereas CSM leverages the inter-block nonuniform access frequency, the intra-block data access features have not been exploited in the MLC design. To be more specific, data patterns within a cache block exhibit two distinct features: (1) The dynamic range is small and usually lies in the lower bits of each word. (2) There is a significant amount of redundancy among the upper-bit parts of words. We refer the first feature as "the static nature" and the second feature as "the repeated nature" of the upper-bit half.

The static nature is caused either by the use of inefficient narrow values or by the nature of computation in some applications (e.g., sparse matrices). Note here a narrow value means a small value saved in a large data type. Typically, the data type needs to be over-provisioned to accommodate

the maximum possible value (i.e., the worst case). While it is very likely that the worse case requires 4 bytes (1 word) but 2 bytes are sufficient to hold the majority of data values. Thus, the lower half of the word is updated much more frequently than the upper half counterpart.

The repeated nature is induced not only by the employment of arrays in representing large data sets, which allocates similar data values and types together in the memory; but also by the the large amount of repeated values at cache level, such as all zeros and ones. Redundancy is widely observed in the data access stream among various applications. For example, zero is used to represent the commonly appeared NULL pointers in programs; a vast amount of adjacent pixels share the same color code in multimedia applications. In fact, cache level data redundancy has been exploited by various data compression schemes to reduce the expensive data movement or data storage.

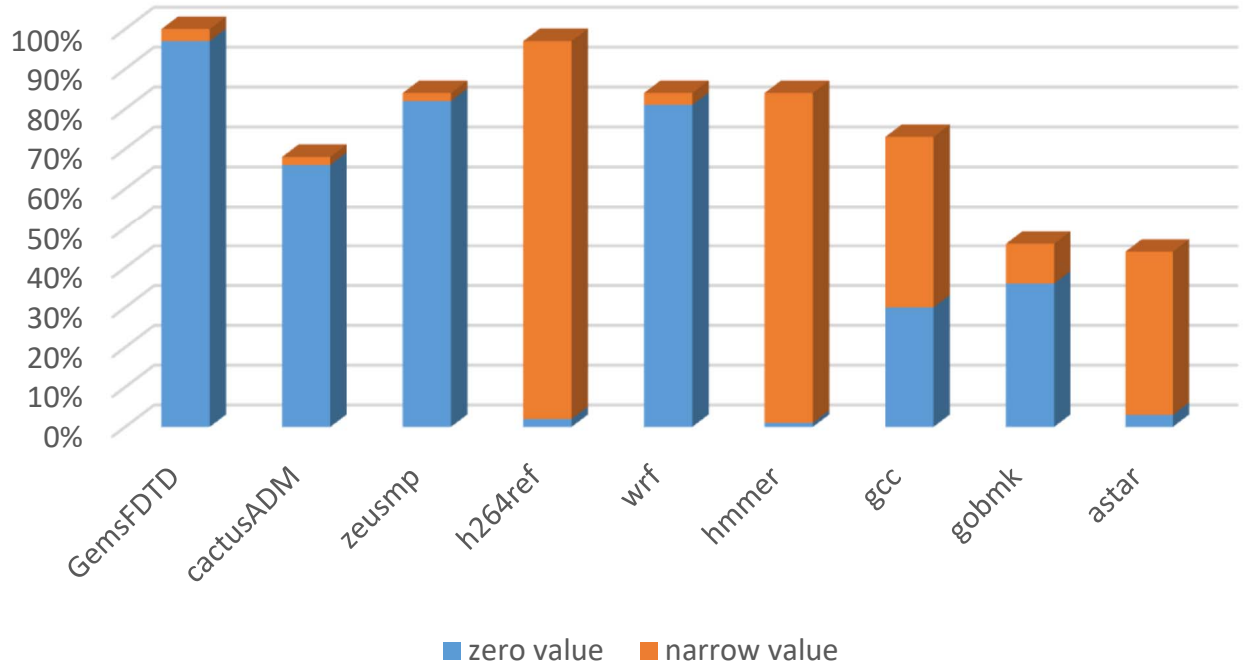

Figure 5.4: The percentage of zero-valued and narrow-valued cache block

As shown in Fig. 5.4, the percentages of zero-valued cache block (exhibits more static nature) and narrow-valued cache block (exhibits more dynamic nature) are evaluated with 9 benchmarks from SPEC in an architecture as described in part IV. On average, 44% cache block belongs to the zero-valued category while 31% cache block exhibit the narrow-valued feature. Actually, this feature has been already leveraged for cache and memory level compression [50][54][41].

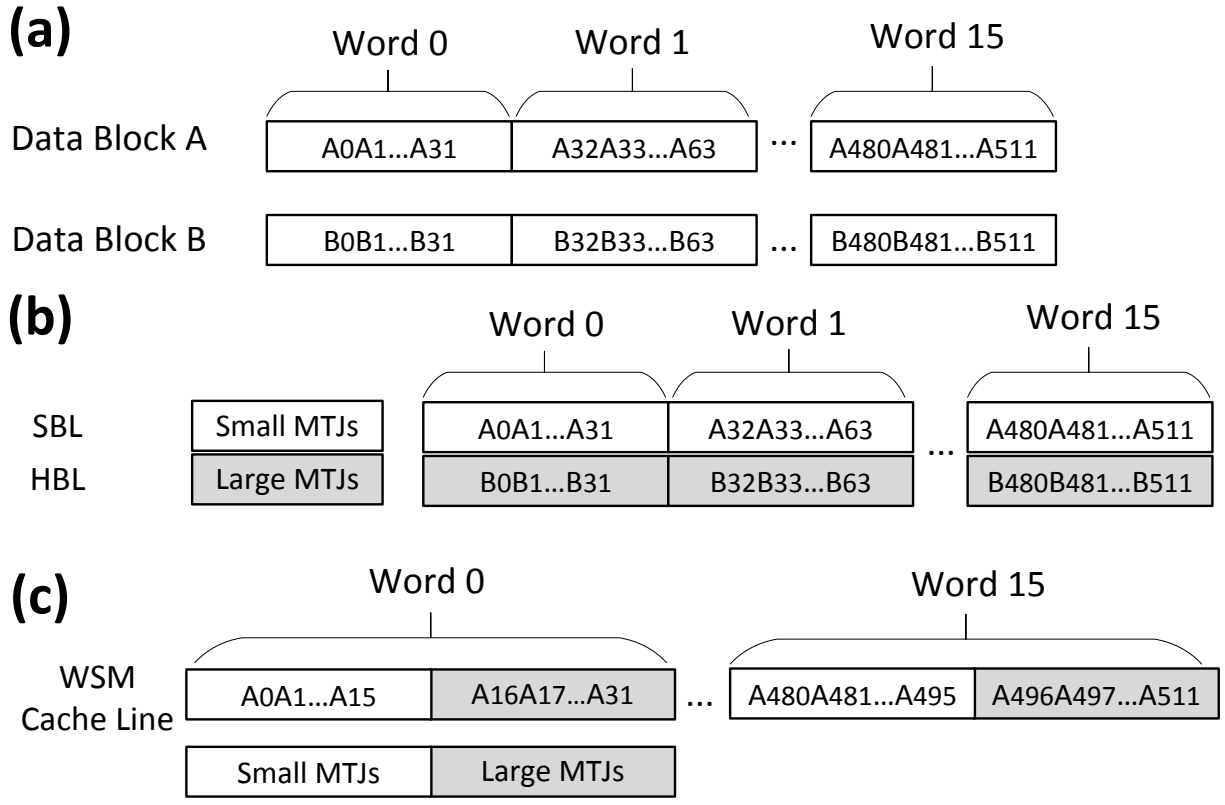

Figure 5.5: (a) 64-byte Data Blocks (b) CSM MLC Design (c) WSM MLC Design

5.3 Double-S Design

5.3.1 Word-Split Mapping

Motivated by the asymmetric bit update behavior within a cache block, we propose an innovative cell-to-block mapping strategy, Word-Split Mapping (WSM), which breaks each word into upper and lower halves. Upper half of the word is mapped to hard bits while the more dynamic lower half is mapped to soft bits. In this paper, we refer "the upper (lower) half of a word in a cache block" as "a upper (lower) half". As shown in Fig. 5.5(c), each 64-byte WSM cache line takes 256 soft bits compared to 512 in a SBL of CSM strategy. Thus, same amount of advantageous soft bits can form more cache lines in WSM. The lower half explicitly benefits from the fact that switching soft bits consumes much less energy. On the other hand, the static nature of the upper half is favorable for hard bits implementation. We further developed the sparse block detection scheme, which leverages the repeated nature of the upper to reduce the write to hard bits. The architecture of MLC STT-RAM based main memory with Double-S mapping strategy is shown in Fig. 5.6.

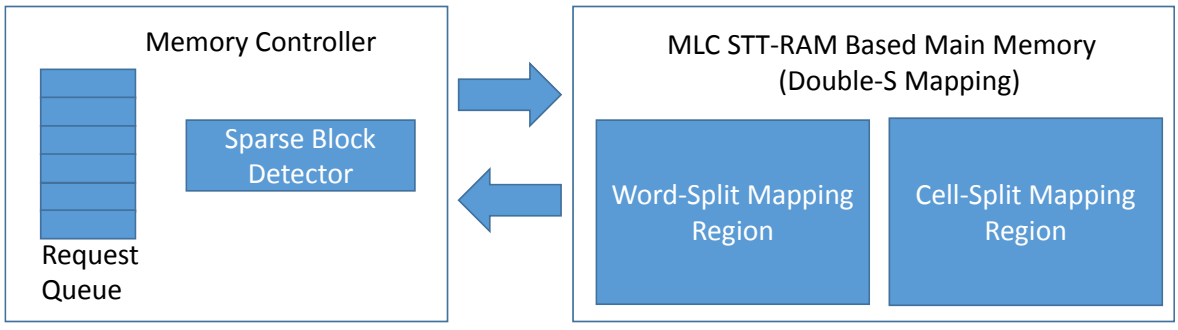

Figure 5.6: MLC STT-RAM Based Main Memory With Double-S Mapping Strategy

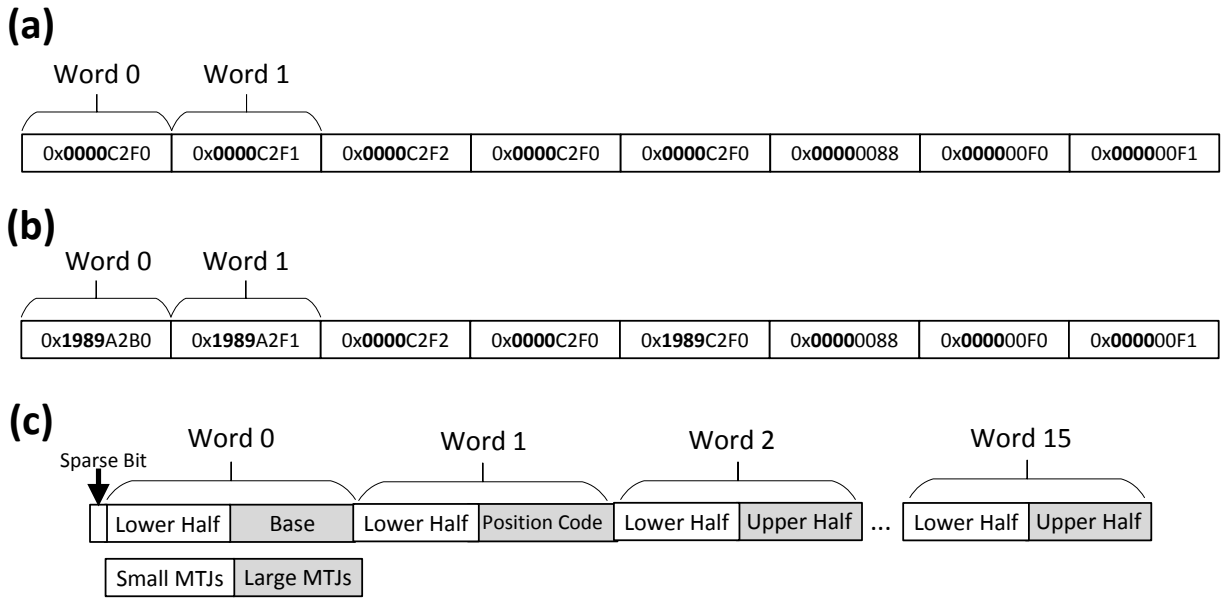

Figure 5.7: (a) A Sparse Block Without Base. (b) A Spares Block With a Base of "1989". (c) Base and Position Code in Upper Halves.

5.3.2 Sparse Cache Block Detection

Before write to the WSM region, a cache block needs to be determined if it is a sparse block or not, which is defined as a block whose upper halves are all zeros, or the remaining non-zero parts are identical. For example, two spare blocks are shown in Fig. 5.7(a) & Fig. 5.7(b). We also refer the common non-zero upper half as "base" here. A cache block can have multiple bases. Ideally, the minimal number of write to upper halves (hard bits) can be achieved by replacing repeated bases with a single replica. However, finding all possible bases in a cache block not only increases the hardware complexity but also introduces significant energy and delay overheads.

Thus, a twofold sparse block detection scheme is developed in this paper. First, OR gates are used to detect if all upper halves are zeros, this is referred to as "Zero Half Detection" as shown in Fig. 5.8(a); Second, for those non-zero upper halves, XOR gates are deployed to check if they are identical or not, this step is referred to as "Identical Base Detection" as shown in Fig. 5.8(b). If a

cache block has no base, all upper halves are ‘0’s, i.e., the output from all OR gates in Fig. 5.8 are ‘0’s. While if a cache block only has one base, the output from the final XOR gate is ‘0’ as shown in Fig. 5.8.

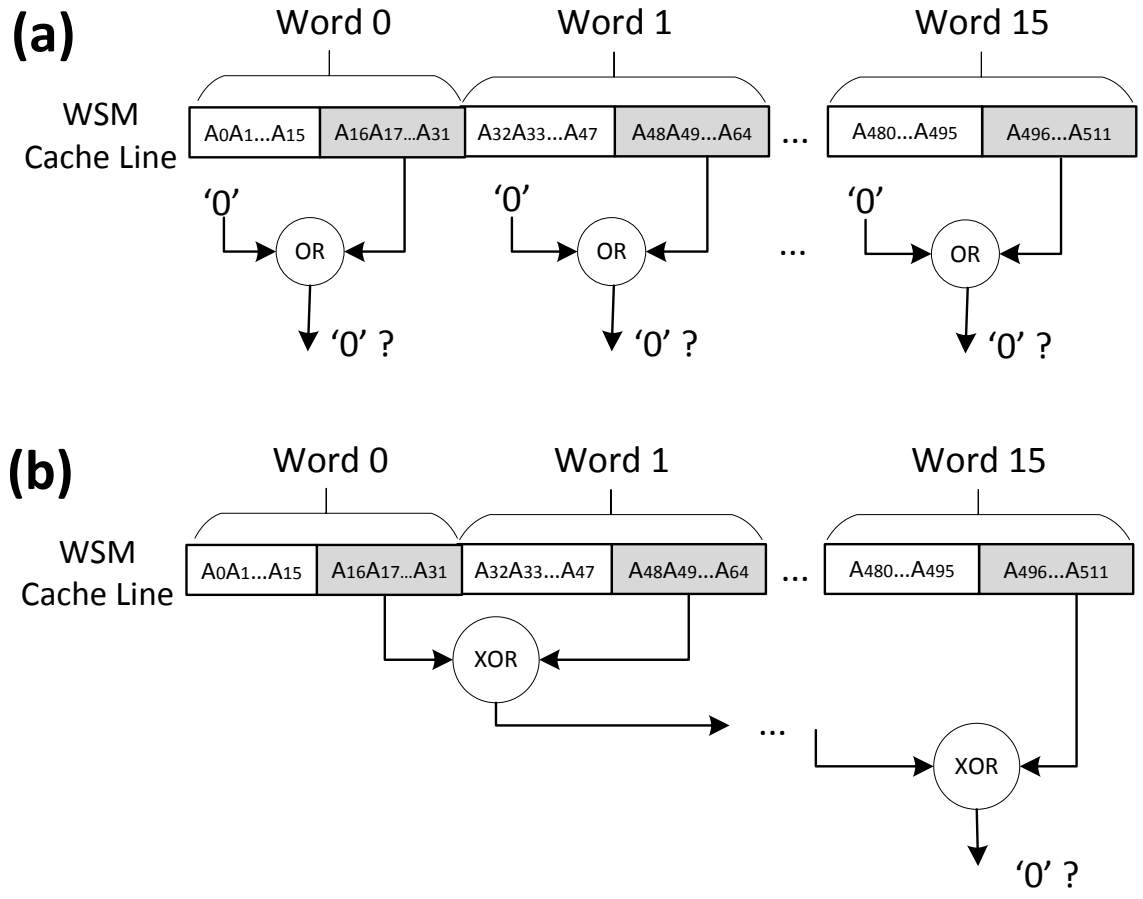

Figure 5.8: (a) Zero Half Detection (b) Identical Base Detection

5.3.3 Write and Read in the WSM Region

In order to minimize the write pressure on hard bits in the WSM region, one more “sparse bit” is attached to each block in the WSM region. The foregoing detection scheme decides the value of this bit. When all upper halves in a cache block are zeros (i.e., the base are all ‘0’s) or these is

only one base, the sparse bit is set to '1' and this cache block is considered as an sparse block. Otherwise, the sparse bit is set to '0'. In addition, we use a simple encoding technique to record the positions of non-zero upper halves. For example, a 64-byte block (16 words) requires a 16-bit long position code to save the location information. For example, if the sparse cache block has no base, the position code is all '0's; if only word 0 and word 1 have an identical base, the position code will be "0000000000000011". Note here the common base is stored in the first upper half, along with the position code saved in the second upper half of the cache block as shown in Fig. 5.7(c).

Thus, write and read in the WSM takes up to two steps. In the write process of a sparse cache block, we first encode the position information, then write lower halves, the base and the position code together. In the end, the sparse bit is marked as '1'. On the other hand, in the read operation, we first read the sparse bit, if it is '1', we decode the position code and read the base, lower halves to recover the cache block. If the sparse bit is '0', the cache block is read as a regular block. The write and read scheme of Doubl-S mapping MLC is shown in Fig. 5.9.

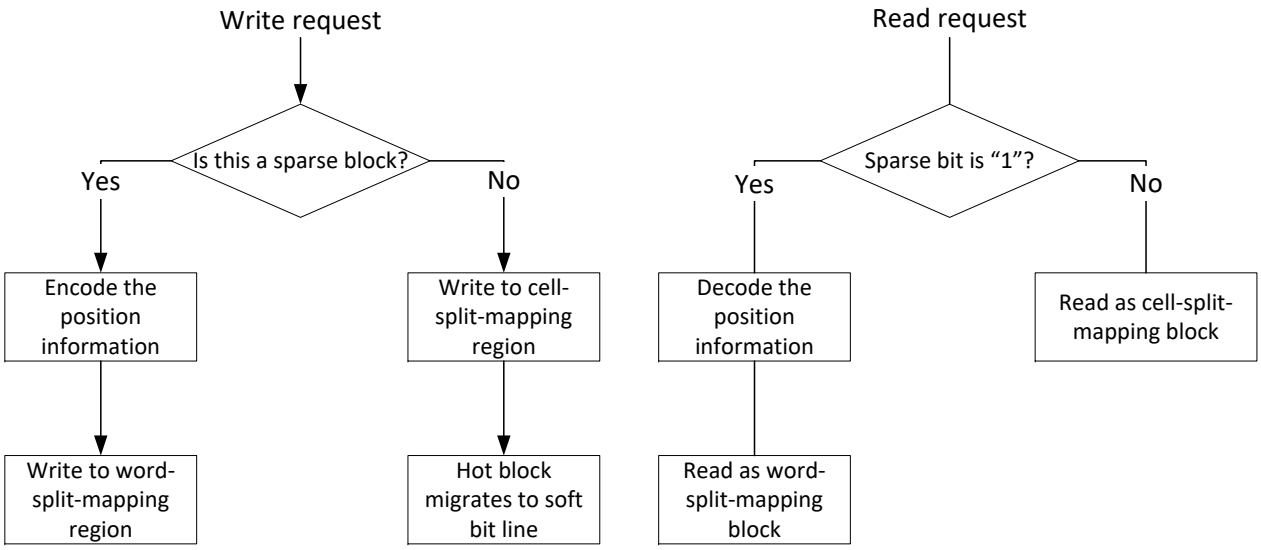

Figure 5.9: Double-S Mapping Write and Read Schemes

5.4 Evaluation

Table 5.1: Baseline Configuration

CPU	4 cores, 3.3 GHz, Fetch/ Exec/ Commit width 4
L1	private, 32 KB, I/D separate, 8-way, 64 B, SRAM, WB
L2	private, 4 MB, unifed, 8-way, 64 B, SRAM, WB
Main Memory	2 GB, MLC STT-RAM, 1 channel, 2 ranks/ channel, 4 bank/ rank

Table 5.2: Comparison of Optimized MLC STT-RAM and DRAM [27][67]

	MLC STT-RAM	DRAM
Read Latency (Cycles)	70	72
Write Latency (Cycles)	160	131
Read Energy	1.03	1.21
Write Energy	2.53	1.21
Row Buffer Access Energy	1	1

The evaluation was conducted by using the cycle-accurate simulator GEM5 [11] connected with DRAMSim2 [61]. We modified cache controller and memory module to realize the proposed function. The full system simulator mimics the computer architecture as shown in Table 5.1. We also show the energy and performance comparison between optimized MLC STT-RAM and DRAM as memory memory in Table 5.2. Benchmarks from SPEC were used for the experiment, executing 500 million instructions starting at the Region Of Interest (ROI) after warming up the cache with 5 million instructions. In this experiment, we use simsmall set as simsmall and simlarge sets will produce similar results.

We adopted the serial MLC STT-RAM cell design from [77]. We used the NVSim and CACTI [27][67] to obtain the key design parameters including read/ write latency and energy consumption for DRAM and MLC STT-RAM. All read and write access energy numbers are normalized

to the row buffer access energy. The MLC STT-RAM adopts both WSM and CSM strategies and comprehensive optimization techniques such as adaptive restore schemes for read and write disturbance.

5.4.1 Energy Consumption

Fig. 5.10 compares the energy consumption including both dynamic and static energy of MLC STT-RAM and DRAM. This energy consumption can be broke down into the following parts: (1) precharge energy consumption which activates a row; (2) read and write operation energy. For DRAM, the row buffer are accessed during both read and write operation. The write operation of DRAM also charge and discharge bitlines and DRAM cells, while MLC STT-RAM write operation does not affect bitline or memory cell; (3) refresh energy for DRAM; (4) restore energy for MLC STT-RAM. When the read disturbance or write disturbance happen, a write back is necessary to keep the data integrity.

DRAM consumes considerably more energy than MLC STT-RAM. This is because the cell refresh operation makes the overall energy consumption enormous. DRAM consumes 90.6% more energy than MLC STT-RAM on average. In some benchmarks like gobmk and hmmer, DRAM consumes up to 100% more energy. The optimization techniques we applied to the MLC STT-RAM includes adaptive restore schemes, dead write bypass, selective and partial write.

5.4.2 Instruction Per Cycle

IPC indicates the overall system performance. Fig. 5.11 compares the IPC of MLC STT-RAM and DRAM with same capacity. The IPC of MLC STT-RAM is decreased by 4.8% on average, while for some benchmarks with less write to the hard bit of MLC, such as hmmer, IPC is only

reduced by 1.8% due to the reduction of longer access latency of HBLs. Although MLC STT-RAM discards a large amount of unnecessary writes, dynamically computing which write can be discarded and comparing it take extra clock cycles.

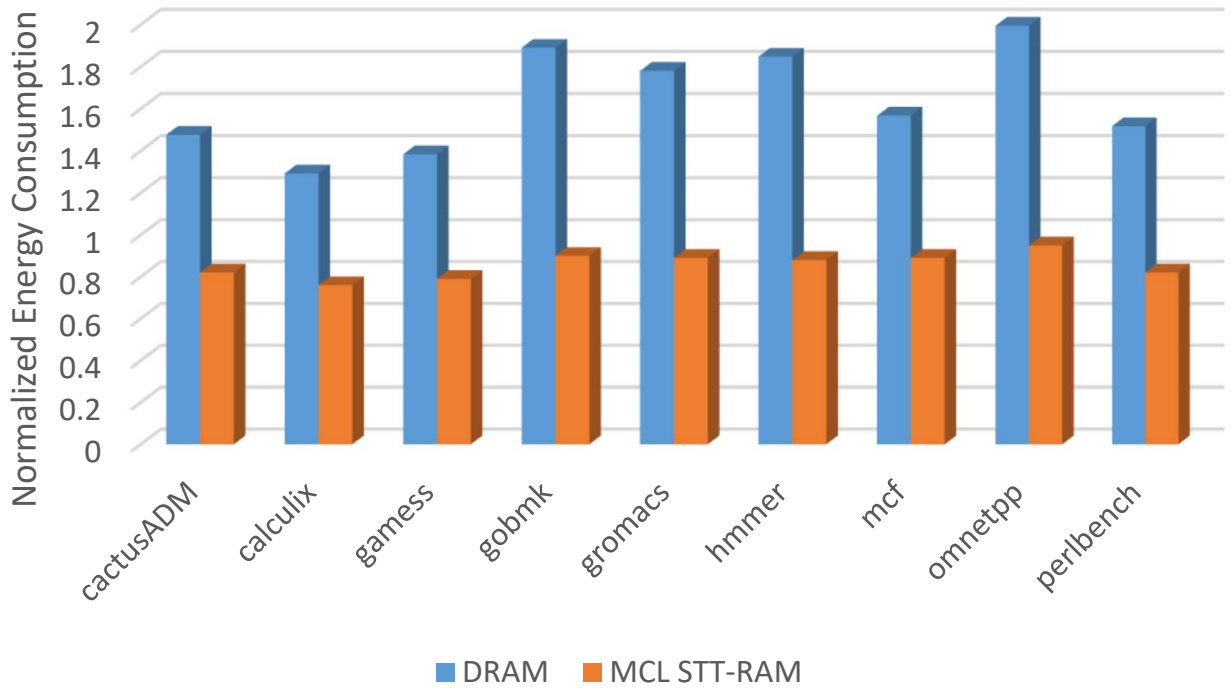

Figure 5.10: Normalized Energy Consumption of MLC STT-RAM and DRAM

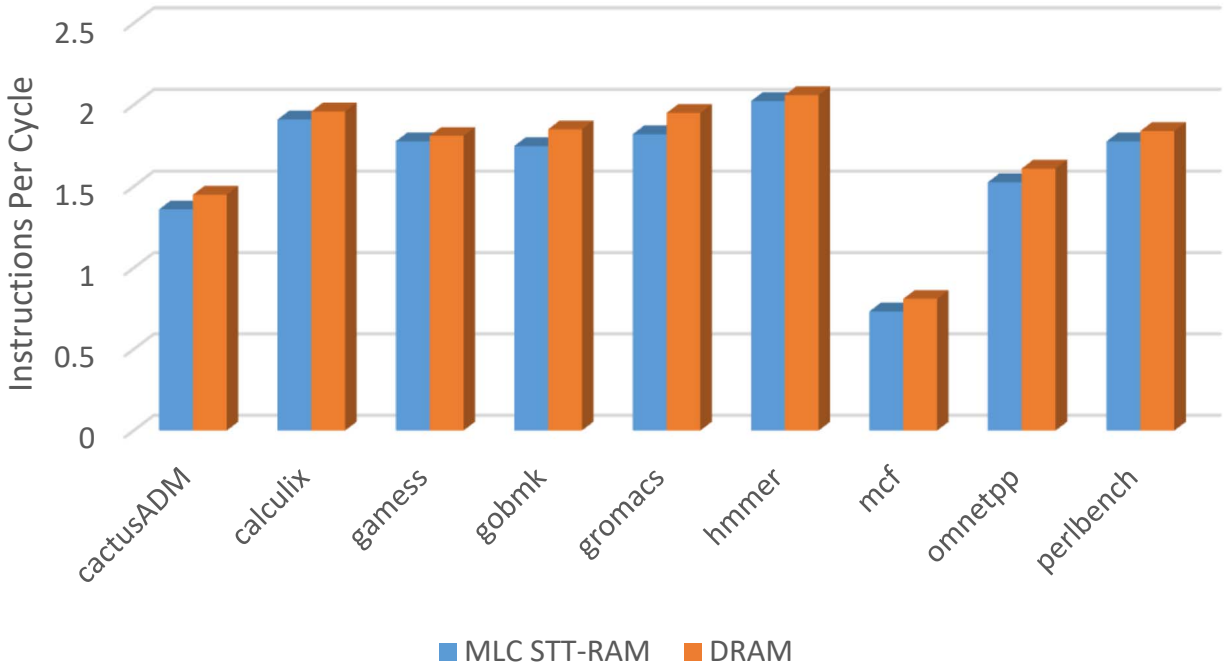

Figure 5.11: System IPC Comparison of MLC STT-RAM and DRAM

CHAPTER 6: RELATED WORK

Many previous work have been done in the area of data management for persistent memory system. We first discuss the well know work about persistent memory from computer system perspective; then we list the work about one of the most promising persistent memory (non-volatile memory), STT-RAM, and its multi-level cell structure; last but not least, we discuss the tradeoff which can be leveraged among performance, reliability and energy-efficiency.

6.1 Persistent Memory System

[25] evaluates the byte-addressable persistent RAM (BPRAM) in terms of file system. The author implemented a specific file system for BPRAM, referred as BPFS. BPFS is much faster than the traditional file systems designed HDD or SSD. Also, BPFS guarantees the durability of file system writes on the persistent store for safety and consistency. BPFS propose to use short-circuit shadow paging. This technique enables copy-on-write at fine granularity. In addition, it atomically commit small changes. This BPFS is different from traditional file system is tailored for persistent memory in many aspects. DRAM buffer is not used for file system, so that the expensive memory space can be saved. BPFS is also designed for random small write. BPFS only write few bytes level of data in place to avoid sending unnecessary traffic data over the memory bus. BPFS greatly mitigates the data vulnerability for those that are not durable.

In [69], the author proposed Mnemosyne, which is a lightweight system to expose persistent memory. A low-level programming interface is provided to get an easy access to the persistent memory. Mnemosyne also simplify the use of persistent memory with three key features: Mnemosyne creates dedicated persistent memory regions, which can be created to hold labeled variables or

allocated dynamically. Mnemosyne also swap storage class memory pages to virtualize persistent regions. In addition, Mnemosyne provides low level operations and interface to support consistent update and a durable memory transaction mechanism to support in-place updates.

Zhao et al. [79] propose a memory control scheme (FIRM) that realize fair memory access and high performance simultaneously. FIRM better utilize the bandwidth for persistent applications and achieve a balanced bandwidth utilization. FIRM first categorizes memory requests into four sources and forms batch requests for each source and treat them fairly. Second, FIRM exploits persistent memory bank level parallelism to maximize the usage of bandwidth. Third, FIRM reschedules the memory access request to minimize bus contention and queue drain.

In [15], Chen et al. tries to integrate persistent memory into computer system with minimal impact to the kernel. To answer the question how the existing operating system can be used to directly access persistent memory as a block device, the author proposed a hybrid model to combine the advantages from both memory based and storage based model. They also implemented a prototype and explored a variety of design trade-offs in terms of performance, protection, persistence and compatibility. The prototype platform shows both protection and ordered persistence can be realized with a small sacrifice of performance.

Lu et al. [48] proposed a novel mechanism which reduces the overhead brought by persistent memory ordering of writes. Different types of persistent memory ordering that impact the performance are identified first. The author found that after some modifications on persistent memory log organization and hardware support, relaxation of persistence ordering can be realized without losing storage consistency requirements. The author introduced a new commit protocol which reduces the usage of commit records. Because of this, the overhead caused by intra-transaction persistence ordering can be eliminated. Another technique proposed enables the speculative update of persistent memory in any order, at the same time, these writes from different transactions can be viewed

by software in program order. Therefore, the overhead of inter-transaction persistence ordering can be reduced.

6.2 Exploration of Multi-Level Cell Design

After MLC STT-RAM was proposed at circuit level [31][77], it has gained tremendous interests as a high-density, low-power and non-volatile memory. Previous research on MLC STT-RAM on-chip cache mainly focused on leveraging the performance disparity of two MTJs. For instance, Jiang et al. [36] proposed to promote frequently written data into write-fast-read-slow soft bit lines, and frequently read data into read-fast-write-slow hard bit lines within parallel MLC structured STT-RAM.

Luo et al. [49] proposed to minimize the two-step transition by comparing with the original code and applying the optimal encoding for the data to be written. As the magnetization direction of two MTJs in a cell can not be switched at the same time. Some state transitions will be done in two steps. The frequent two step transition will wear out the soft bit gradually and impact the lifetime of MLC. Luo et al. employed different encoding for the incoming data. Depends on how many transitions is required, the proposed scheme will adopt the optimal encoding. Whereas the foregoing work utilize the parallel MLC model, there are several research work adopting the serial MLC.

Bi et al. [9] advocated a bit mapping strategy constructing general fast- and slow- lines which is used in this paper. Wang et al. [70] proposed to dynamically disable the hard bits in cache line while keep the number of associativity. Chi et al. [23] presented an efficient local checkpointing method by storing working data in soft bits and checkpoint data in hard bits. MLC is able to store both working data and checkpoint data in the same cell, therefore it becomes an ideal technology

for local checkpoint. Compared to the traditional checkpoint method which moves data from backup storage or main disk to memory, MLC based method transfers data within the cell and avoid expensive data movement cost. The author adopts MLC STT-RAM instead of MLC PCM as the main memory, because writing in PCM comes with program-and-verify process and incurs significant latency and energy overhead. In contrast, writing in MLC STT-RAM takes up to two steps, so it is faster and more energy efficient than that in PCM.

In [47], Liu et al. proposed to use data compression technique to maximize the use of the advantageous soft bit. With data compression, cache lines can be compressed into half of the original size. The compressed cache line can be fit into the soft bit line directly, which reduces the slow and power-consuming hard bit line accesses. Two techniques are proposed, the first one is overhead optimization and the second one is leveraging the unused hard bit lines to store more cache blocks. Liu et al. adopted interleaved mapping in their research. Instead of using dynamic block size, their work fits compressible line into the advantageous soft bit region first.

In [78], Zhao et al. implemented main memory using MLC STT-RAM and tried to realize energy-efficient data movement with it for image processing applications. This design includes two parts. The first one supports approximate image memory by writing the an approximate image into the soft bit region of MLC STT-RAM main memory. Because of the asymmetric write current of MLC, writing to soft bit only will trade precision for energy efficiency. The software interface with image applications will determine the level of acceptable approximate image. The second part is an approximate mode controller which is integrated memory controller. This controller dynamically schedule and coordinate precise and approximate writes and read access to the main memory. However, this design unavoidably incurs additional hardware requirement and software interface.

Similar to the previous work from Zhao et al., [63] introduced approximation-aware MLC STT-

RAM architecture and management policies. This architecture employs three core units design. The first one is a latency aware error correction unit, the critical delay is eliminated to maintain the performance while a double-error correction algorithm is developed to protect data correctness. The second one is the cache management policy for approximate MLC. Cache sets are categorized as reliable and unreliable, then the unreliable sets can be used to store approximate data without enforcing strict error correction. By doing this, the author trade reliability for performance and energy efficiency. The third unit is application-aware quality control. This unit dynamically adapt and adjust the level of approximation in storage by monitoring the application's output quality.

6.3 Performance, Reliability and Energy Efficiency Trade-off

Meanwhile, the reliability issue of STT-RAM is gaining significant research interest. Recently, Wang et al. [71] proposed a selective restore method to mitigate the overhead brought by read disturbance correction in SLC STT-RAM. They also added three more flag bits in each cache line to assist their scheme. Our work shares some similarities with this one, but our research also address the write disturbance issue in MLC.

Jiang et al. [35] also approached the read disturbance issue by adaptively selecting between High Current Restore Required (HCRR) reads and Long Current Long Latency (LCLL) reads. Our previous work [18] presented ARS-WD to correct write disturbance in MLC with low energy consumption overhead, however, the read disturbance issue of soft bit was not discussed. In [29], ternary coding technique is proposed to remove the most error-prone state and trade MLC capacity for reliability. Because of process variation and thermal fluctuations, the read and write of STT-RAM can be unstable at nanometer technology nodes. This has a serious impact on MLC STT-RAM since it use a limited range of resistance value to represent data. After the capacity is traded for stability, less amount of data is stored in a cell, but ternary cache can achieve higher stability for

read. Ternary cache also adopt novel write mechanism to improve the reliability. A high current is induced to switch the MTJ cell. At the same time, the resistance state is carefully monitored and write current is terminated right after the data is successfully written.

Wen et al. [73] considered the nonuniform ECC design requirement for MLC and advocated a tri-region mapping strategy to reduce the write pressure on hard bit lines. The tri-region MLC STT-RAM can meet the performance, energy, and reliability requirement at the same time. This work points out that the widely adopted CSM has reliability issue. Cache lines are classified into three types: soft lines, hard lines and mixed lines. The cache data will be assigned and dynamically stored into the corresponding cache line based on the PER feature. Non-uniform strength ECC is also proposed to ensure the data correctness for different types of cache lines. For example, for the more reliable cache line, simple ECC can be adopt to trade ECC overhead for performance.

Li et al. [46] proposed a compiler-assisted solution to relax the non-volatility of STT-RAM while minimize the refresh operations. Since non-volatility can be relaxed to trade for write performance, a refresh scheme is required to prevent data from losing. The author propose to reschedule the memory access sequence by manipulating data layout. The total number of active fresh will be reduced by changing the feature of passive refresh. An integer linear programming solution and a heuristic algorithm are proposed to solve the active fresh minimization problem. In the case the interval between two memory access are too long to handle, the author propose a threshold number of times a data block can be refreshed. When the refresh operations is reduced, the energy efficiency will be improved.

In [74], Wen et al. performed quantitative evaluation to show the challenges of employing simple ECC for MLC STT-RAM to realize acceptable reliability and yield. Then state-restrict MLC STT-RAM is proposed to ensure the data integrity and enhance the performance. At the circuit level, state restriction, error pattern removal and ternary coding to eliminate the read and write error. At

architecture level, state pre-recovery will help to reduce the unnecessary two step write access and average cache access time.

CHAPTER 7: CONCLUSION AND FUTURE WORK

In this section, we summarize our work in this dissertation and list potential future direction of the research:

Firstly, we proposed ARS-WD to mitigate the energy overhead caused by WD restores in MLC STT-RAM. We first introduce the concept of RRD, which exploits the number of memory access, to quantify the timing distance between two successive reads to the same cache block. We also developed an RRD predictor consisting of a read sampler and RRD prediction table to provide trustworthy RRD for each cache block. Upon a write request to an HBL, ARS-WD compares the EDR of corresponding SBL with a threshold value to determine if restores can be postponed or not. We conducted memory access analysis to adopt a threshold value with high confidence.

Secondly, we proposed ARS-RD to correct the soft bit read disturbance issue. ARS-RD delays the restoration till the eviction from higher level cache, so that a significant amount of restore operation can be merged with dirty block write-back. We gave a detailed example showing how read sampler and read reuse prediction table work. Since we make an assumption that the multi-level cache hierarchy adopts non-inclusive model, we discussed the necessary modification to be made to enable our ARS-WD and ARS-RD on inclusive and exclusive models. Our experimental results show that ARS effectively reduces both dynamic and overall system energy dissipation, significantly decreases EAT product and improves the IPC with negligible storage overhead.

Thirdly, we explored and identified the possibilities and challenges of employing MLC STT-RAM in the persistent main memory system. To minimize the use of expensive hard bits in MLC, we proposed a Doulbe-S mapping strategy, combining the widely used cell-split-mapping with the novel word-split-mapping strategy, that exploit the intra block data access feature. We perform comprehensive experiment to mimic the MLC STT-RAM main memory parameter and architecture.

The results show that MLC can significantly reduce the overall energy consumption but trade-off a certain degree of IPC for the optimization techniques used in MLC.

To further extend our research, there are several future direction we can pursue. First, ARS-WD and ARS-RD are proposed based on few assumptions: cell split mapping is the only mapping strategy used and non-inclusive property is enforced. To apply ARS on MLC with other mapping strategy, a holistic data migration and scheduling policy is desired. Second, whereas our research deploy MLC as the main memory replacement, the current capacity of MLC can barely achieve gigabyte level. Thus, it is still necessary introduce DRAM for large and cost-effective memory system. We will explore the memory scheduling and partition problems of DRAM and MLC hybrid system in the future.

LIST OF REFERENCES

[1] Intel Core i7 Processor. In *http://www.intel.com/content/www/us/en /processors/core/core-i7-processor.html*.

[2] Predictive Technology Model (PTM) . In *http://ptm.asu.edu/*.

[3] A. Agrawal, A. Ansari, and J. Torrellas. Mosaic: Exploiting the spatial locality of process variation to reduce refresh energy in on-chip edram modules. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, 2014.

[4] A. Agrawal, P. Jain, A. Ansari, and J. Torrellas. Refrint: Intelligent refresh to minimize power in on-chip multiprocessor cache hierarchies. In *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, 2013.

[5] J. Ahn, S. Yoo, and K. Choi. DASCA: Dead Write Prediction Assisted STT-RAM Cache Architecture. In *20th International Symposium on High Performance Computer Architecture, HPCA*. IEEE.

[6] J. Ahn, S. Yoo, and K. Choi. Prediction Hybrid Cache: An Energy-Efficient STT-RAM Cache Architecture. In *IEEE Transactions on Computers*, 2016.

[7] B. Bass. A low-power, High-performance, 1024-Point FFT Processor. In *IEEE Journal of Solid-State Circuits*, 1999.

[8] L. A. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Systems Journal*, 5(2):78–101, 1966.

[9] X. Bi, M. Mao, D. Wang, and H. Li. Unleashing the Potential of MLC STT-RAM Caches. In *Proceedings of the International Conference on Computer-Aided Design, ICCAD*. IEEE Press, 2013.

[10] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The Parsec Benchmark Suit: Characterization and Architectural Implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques, PACT*. ACM, 2008.

[11] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2), Aug.

[12] Y. Cai, Y. Luo, S. Ghose, and O. Mutlu. Read disturb errors in mlc nand flash memory: Characterization, mitigation, and recovery. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2015.

[13] Y. Cai, Y. Luo, E. F. Haratsch, K. Mai, and O. Mutlu. Data retention in mlc nand flash memory: Characterization, optimization, and recovery. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015.

[14] M. T. Chang, P. Rosenfeld, S. L. Lu, and B. Jacob. Technology comparison for large last-level caches (l3cs): Low-leakage sram, low write-energy stt-ram, and refresh-optimized edram.

In *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, 2013.

[15] F. Chen, M. P. Mesnier, and S. Hahn. A protected block device for persistent memory. In *2014 30th Symposium on Mass Storage Systems and Technologies (MSST)*, June 2014.

[16] X. Chen, N. Khoshavi, R. F. DeMara, J. Wang, D. Huang, W. Wen, and Y. Chen. Energy-aware adaptive restore schemes for mlc stt-ram cache. *IEEE Transactions on Computers*, 66(5):786–798, 2017.

[17] X. Chen, N. Khoshavi, J. Zhou, D. Huang, R. F. DeMara, J. Wang, W. Wen, and Y. Chen. Aos: Adaptive overwrite scheme for energy-efficient mlc stt-ram cache. In *Proceedings of the 53rd Annual Design Automation Conference*, DAC '16.

[18] X. Chen, N. Khoshavi, J. Zhou, D. Huang, R. F. DeMara, J. Wang, W. Wen, and Y. Chen. AOS: Adaptive Overwrite Scheme for Energy Efficient MLC STT-RAM Cache. In *53nd Annual Design Automation Conference, DAC*. ACM.

[19] Y. Chen, H. H. Li, I. Bayram, and E. Eken. Recent technology advances of emerging memories. *IEEE Design Test*, 34(3):8–22, 2017.

[20] Y. Chen, X. Wang, W. Zhu, H. Li, Z. Sun, G. Sun, and Y. Xie. Access Scheme of Multi-level Cell Spin-transfer Torque Random Access Memory and its Optimization. In *53rd IEEE International Midwest Symposium on Circuits and Systems*, 2010.

[21] Y. Chen, W. F. Wong, H. Li, C. K. Koh, Y. Zhang, and W. Wen. On-chip Caches Built on Multilevel Spin-transfer Torque RAM Cells and Its Optimizations. In *Journal on Emerging Technologies in Computing*. ACM.

[22] H. Y. Cheng, J. Zhao, J. Sampon, M. J. Irwin, A. Jaleel, Y. Lu, and Y. Xie. LAP: Loop-Block Aware Inclusion Properties for Energy-Efficient Asymmetric Last Level Caches. In *43th International Symposium on Computer Architecture, ISCA*. IEEE, 2016.

[23] P. Chi, C. Xu, T. Zhang, X. Dong, and Y. Xie. Using Multi-Level Cell STT-RAM for Fast and Energy-Efficient Local Checkpointing. In *IEEE/ACM International Conference on Computer-Aided Design, ICCAD*. IEEE Press, 2014.

[24] K. C. Chun, H. Zhao, J. D. Harms, T. H. Kim, J. P. Wang, and C. H. Kim. A Scaling Roadmap and Performance Evalution of In-Plane and Perpendicular MTJ Based STT-MRAMs for High-Density Cache Memory. In *IEEE Journal of Solid-State Circuits*. IEEE Press, 2013.

[25] J. Condit, E. B. Nightingale, C. Frost, E. Ipek, B. Lee, D. Burger, and D. Coetzee. Better i/o through byte-addressable, persistent memory. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, SOSP '09, New York, NY, USA, 2009. ACM.

[26] G. Dhiman, R. Ayoub, and T. Rosing. Pdram: A hybrid pram and dram main memory system. In *2009 46th ACM/IEEE Design Automation Conference*.

[27] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi. NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, July 2012.

[28] M. Elver and V. Nagarajan. Tso-cc: Consistency directed cache coherence for tso. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, 2014.

[29] S. Hong, J. Lee, and S. Kim. Ternary Cache: Three-valued MLC STT-RAM Caches. In *32nd International Conference on Computer Design, ICCD*. IEEE Press, 2014.

[30] M. Imani, S. Patil, and T. Rosing. Low Power Data Aware STT-RAM based Hybrid Cache Architecture. In *17th International Symposium on Quality Electronic Design, ISQED*. IEEE, 2016.

[31] T. Ishigaki, T. Kawahara, R. Takemura, K. Ono, K. Ito, H. Matsuoka, and H. Ohno. A Multi-level-cell Spin-transfer Torque Memory with Series-stacked Magnetotunnel Junctions. In *2010 Symposium on VLSI Technology*. IEEE.

[32] T. Ishigaki, T. Kawahara, R. Takemura, K. Ono, K. Ito, H. Matsuoka, and H. Ohno. A Multi-Level Cell Spin-Transfer Torque Memory With Series-Stacked Magnetotunnel Junctions. In *In Proceedings of the Symposium on VLSI Technology*, 2010.

[33] A. Jaleel, E. Borch, M. Bhandaru, S. C. S. Jr., and J. Emer. Achieving Non-Inclusive Cache Performance with Inclusive Caches: Temporal Locality Aware (TLA) Cache Management Policies. In *43rd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO*, 2010.

[34] A. Jaleel, K. B. Theobald, S. C. S. Jr, and J. Emer. High performance cache replacement using Re-Reference Interval Prediction (RRIP). In *Proceedings of the 37th Annual International Symposium on Computer Architecture, ISCA*. ACM, 2010.

[35] L. Jiang, W. Wen, D. Wang, and L. Duan. Improving read performance of STT-MRAM based main memories through Smash Read and Flexible Read. In *IEEE 21st Asia and South Pacific Design Automation Conference, ASP-DAC*, 2016.

[36] L. Jiang, B. Zhao, Y. Zhang, and J. Wang. Constructing Large and Fast Multi-Level Cell STT-MRAM Based Cache for Embedded Processors. In *49th ACM/EDAC/IEEE Design Automation Conference, DAC*. IEEE Press, 2012.

[37] A. Jog, A. K. Mishra, C. Xu, Y. Xie, V. Narayanan, R. Iyer, and C. R. Das. Cache Revive: Architecting Volatile STT-RAM Caches for Enhanced Performance in CMPs. In *49th ACM/EDAC/IEEE Design Automation Conference, DAC*. IEEE Press, 2012.

[38] S. H. Kang. Embedded STT-MRAM for Mobile Application: Enabling Advanced Chip Architectures. In *Non-Volatile Memories Workshop*, 2010.

[39] G. Keramidas, P. Petoumenos, and S. Kaxiras. Cache Replacement Based on Reuse-Distance Prediction. In *25th International Conference on Computer Design*. IEEE Press, 2007.

[40] N. Khoshavi, X. Chen, J. Wang, and R. F. DeMara. Read-Tuned STT-RAM and eDRAM Cache Hierarchies for Throughput and Energy Enhancement. 2016.

[41] N. S. Kim, T. Austin, and T. Mudge. Low-energy data cache using sign compression and cache line bisection. In *Proceedings of the Workshop on Memory Performance Issues*, May 2002.

[42] E. Kltrsay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu. Evaluating stt-ram as an energy-efficient main memory alternative. In *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2013.

[43] A.-C. Lai and B. Falsafi. Selective, accurate, and timely self-invalidation using last-touch prediction. 2000.

[44] A.-C. Lai, C. Fide, and B. Falsafi. Dead-block prediction & dead-block correlating prefetchers. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, ISCA '01. ACM, 2001.

[45] L. Li, D. Tong, Z. Xie, J. Lu, and X. Cheng. Optimal bypass monitor for high performance last-level caches. In *2012 21st International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2012.

[46] Q. Li, Y. He, J. Li, L. Shi, Y. Chen, and C. J. Xue. Compiler-Assisted Refresh Minimization for Volatile STT-RAM Cache. In *IEEE Transactions on Computers*, 2015.

[47] L. Liu, P. Chi, S. Li, Y. Cheng, and Y. Xie. Building energy-efficient multi-level cell stt-ram caches with data compression. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2017.

[48] Y. Lu, J. Shu, L. Sun, and O. Mutlu. Loose-ordering consistency for persistent memory. In *2014 IEEE 32nd International Conference on Computer Design (ICCD)*, Oct 2014.

[49] H. Luo, J. Hu, L. Shi, C. J. Xue, and Q. Zhuge. Two-Step State Transition Minimization for Lifetime and Performance Improvement on MLC STT-RAM. In *53nd Annual Design Automation Conference, DAC*. ACM.

[50] S. Mittal and J. S. Vetter. A survey of architectural approaches for data compression in cache and main memory systems. *IEEE Trans. Parallel Distrib. Syst.*, 27(5):1524–1536, May 2016.

[51] S. Mittal, J. S. Vetter, and L. Jiang. Addressing read-disturbance issue in stt-ram by data compression and selective duplication. *IEEE Computer Architecture Letters*, 2017.

[52] K. Ono, K. M. T. Kawahara, R. Takemura, J. H. K. I. H. T. S. I. H. H. H. M. H. Yamamoto, M. Yamanouchi, and H. Ohno. A Disturbance-free Read Scheme and a Compact Stochastic-Spin-Dynamics-Based MTJ Circuit Model for Gb-Scale SPRAM. In *IEEE International Electron Devices Meeting, IEDM*, 2009.

[53] A. Patel, F. Afram, S. Chen, and K. Ghose. MARSS: a Full System Simulator for multicore x86 CPUs. In *48th ACM/EDAC/IEEE Design Automation Conference, DAC*. IEEE Press, 2011.

[54] G. Pekhimenko, V. Seshadri, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry. Base-delta-immediate compression: Practical data compression for on-chip caches. In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*, PACT '12. ACM, 2012.

[55] P. Petoumenos, G. Keramidas, and S. Kaxiras. Instruction-based Reuse-distance Prediction for Effective Cache Management. In *Proceedings of the 9th International Conference on Systems, Architectures, Modeling and Simulation, SAMOS*. IEEE Press, 2009.

[56] P. Petoumenos, G. Keramidas, and S. Kaxiras. Instruction-based Reuse Distance Prediction Replacement Policy. In *1st JILP Workshop on Computer Architecture Competitions (JWAC-1) in conjunction with ISCA-37*, 2010.

[57] M. D. Powell, A. Agarwal, T. N. Vijaykumar, B. Falsafi, and K. Roy. Reducing set-associative cache energy via way-prediction and selective direct-mapping. In *Proceedings. 34th ACM/IEEE International Symposium on Microarchitecture. MICRO-34*, 2001.

[58] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, and J. Emer. Adaptive insertion policies for high performance caching. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ISCA '07, 2007.

[59] B. P. Railing, E. R. Hein, and T. M. Conte. Contech: Efficiently generating dynamic task graphs for arbitrary parallel programs. *ACM Trans. Archit. Code Optim.*, 12(2), Jul 2015.

[60] L. E. Ramos, E. Gorbatov, and R. Bianchini. Page placement in hybrid memory systems. In *Proceedings of the International Conference on Supercomputing*, ICS '11.

[61] P. Rosenfeld, E. Cooper-Balis, and B. Jacob. Dramsim2: A cycle accurate memory system simulator. *IEEE Comput. Archit. Lett.*, 10(1).

[62] S. Khan and A. R. Alameldeen and C. Wilerson and O. Mutlu and D. A. Jimenezz. Improving cache performance using read-write partitioning. In *IEEE 20th International Symposium on High Performance Computer Architecture, HPCA*, 2014.

[63] F. Sampaio, M. Shafique, B. Zatt, S. Bampi, and J. Henkel. Approximation-aware multi-level cells stt-ram cache architecture. In *Proceedings of the 2015 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, CASES '15, 2015.

[64] V. Seshadri, S. Yedkar, H. Xin, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry. Mitigating prefetcher-caused pollution using informed caching policies for prefetched blocks. *ACM Trans. Archit. Code Optim.*, 11(4), Jan.

[65] J. Sim, J. Lee, M. K. Qureshi, and H. Kim. FLEXclusion: Balancing Cache Capacity and On-chip Bandwidth via Flexible Exclusion. In *39th International Symposium on Computer Architecture, ISCA*. IEEE, 2012.

[66] R. Takemura, T. Kawahara, K. Ono, K. Miura, H. Matsuoka, and H. Ohno. Highly-Scalable Disruptive Reading Scheme for Gb-Scale SPRAM And Beyond. In *IEEE International Memory Workshop (IMW)*, 2010.

[67] S. Thoziyoor, J. H. Ahn, M. Monchiero, J. B. Brockman, and N. P. Jouppi. A Comprehensive Memory Modeling Tool and its Application to the Design and Analysis of Future Memory Hierarchies. In *35th International Symposium on Computer Architecture, ISCA*. IEEE Press, 2008.

[68] R. Venkatesan, V. J. Kozhikkottu, M. Sharad, C. Augustine, A. Raychowdhury, K. Roy, and A. Raghunathan. Cache Design with Domain Wall Memory. In *IEEE Transactions on Computers*, 2015.

[69] H. Volos, A. J. Tack, and M. M. Swift. Mnemosyne: Lightweight persistent memory. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVI, New York, NY, USA, 2011. ACM.

[70] J. Wang, P. Roy, W. F. Wong, X. Bi, and H. Li. Optimizing MLC-based STT-RAM Caches by Dynamic Block Size Reconfiguration. In *IEEE 32nd International Conference on Computer Design, ICCD*, 2014.

[71] R. Wang, L. Jiang, Y. Zhang, L. Wang, and J. Yang. Selective Restore: an Energy Efficient Read Disturbance Mitigation Scheme for Future STT-MRAM. In *52nd ACM/EDAC/IEEE Design Automation Conference, DAC*. IEEE Press, 2015.

[72] Z. Wang, D. A. Jiménez, C. Xu, G. Sun, and Y. Xie. Adaptive Placement and Migration Policy for an STT-RAM-Based Hybrid Cache. In *20th International Symposium on High Performance Computer Architecture, HPCA*. IEEE, 2014.

[73] W. Wen, M. Mao, H. Li, Y. Chen, Y. Pei, and N. Ge. A Holistic Tri-region MLC STT-RAM Design with Combined Performance, Energy, and Reliability Optimizations. In *Design, Automation and Test in Europe Conference and Exhibition, DATE*. IEEE Press, 2016.

[74] W. Wen, Y. Zhang, M. Mao, and Y. Chen. State-Restrict MLC STT-RAM Designs for High-Reliable High-Performance Memory System. In *51st ACM/EDAC/IEEE Design Automation Conference, DAC*. IEEE Press, 2014.

[75] C. Wilkerson, A. R. Alameldeen, Z. Chishti, W. Wu, D. Somasekhar, and S.-L. Lu. Reducing Cache Power with Low-cost, Multi-bit Error-correcting Codes. In *37th annual International Symposium on Computer architecture, ISCA*. IEEE Press, 2010.

[76] H. Yoon, J. Meza, R. Ausavarungnirun, R. A. Harding, and O. Mutlu. Row buffer locality aware caching policies for hybrid memories. In *2012 IEEE 30th International Conference on Computer Design (ICCD)*, 2012.

[77] Y. Zhang, L. Zhang, W. Wen, G. Sun, and Y. Chen. Multi-level Cell STT-RAM: Is It Realistic or Just a Dream. In *IEEE/ACM International Conference on Computer-Aided Design, ICCAD*. IEEE Press, 2012.

[78] H. Zhao, L. Xue, P. Chi, and J. Zhao. Approximate Image Storage with Multi-level Cell STT-MRAM Main Memory. In *IEEE/ACM International Conference on Computer-Aided Design, ICCAD*. IEEE Press, 2017.

[79] J. Zhao, O. Mutlu, and Y. Xie. Firm: Fair and high-performance memory control for persistent memory systems. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec 2014.

[80] W. Zhao, C. Chappert, V. Javerliac, and J.-P. Nozière. High Speed, High Stability and Low Power Sensing Amplifier for MTJ/CMOS Hybrid Logic Circuits. In *IEEE Transactions on Magnetics*, 2009.