INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI

LiNoVo: Longevity Enhancement of Non-Volatile Caches by Placement, Write-Restriction & Victim Caching in Chip Multi-Processors

by

Sukarn Agarwal

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

in the

Department of Computer Science and Engineering

Under the supervision of

Prof. Hemangee K. Kapoor

March 2020

भारतीय प्रौद्योगिकी संस्थान गुवाहाटी
Indian Institute of Technology Guwahati

Declaration of Authorship

I, Sukarn Agarwal, hereby confirm that:

- The work contained in this thesis is original and has been done by myself under the general supervision of my supervisor.
- This work has not been submitted to any other Institute for any degree or diploma.
- Whenever I have used materials (data, theoretical analysis, results) from other sources, I have given due credit to the authors/researchers by citing them in the text of the thesis and giving their details in the reference.
- Whenever I have quoted from the work of others, the source is always given.

Sukarn Agarwal
Research Scholar,
Department of CSE,
Indian Institute of Technology Guwahati,
Guwahati, Assam, INDIA 781039,
sukarn@iitg.ac.in, sukarnagarwal@gmail.com

Date: March 13, 2020
Place: IIT Guwahati

भारतीय प्रौद्योगिकी संस्थान गुवाहाटी
Indian Institute of Technology Guwahati

Certificate

This is to certify that the thesis entitled **"LiNoVo: Longevity Enhancement of Non-Volatile Caches by Placement, Write-Restriction & Victim Caching in Chip Multi-Processors"** being submitted by **Mr. Sukarn Agarwal** to the department of *Computer science and Engineering, Indian Institute of Technology Guwahati*, is a record of bonafide research work under my supervision and is worthy of consideration for the award of the degree of Doctor of Philosophy of the Institute.

Prof. Hemangee K. Kapoor
Department of CSE,
Indian Institute of Technology Guwahati,
Guwahati, Assam, INDIA 781039,
hemangee@iitg.ac.in

Date: March 13, 2020
Place: IIT Guwahati

भारतीय प्रौद्योगिकी संस्थान गुवाहाटी
Indian Institute of Technology Guwahati

Dedicated to

Almighty KRISHNA

and

my loving parents & sisters.

भारतीय प्रौद्योगिकी संस्थान गुवाहाटी
Indian Institute of Technology Guwahati

Acknowledgements

It is an immense pleasure for me to thank all the peoples who have supported me during my Ph.D. stay at IIT Guwahati. First and foremost, I thank my supervisor Prof. Hemangee K. Kapoor for her guidance, encouragement, and extensive help over the last four years. She has given me the freedom to pursue research ideas and develop my research skills. I profusely thank her for correcting my silly mistakes again and again and keep me engaged on Ph.D. work.

I am thankful to my Doctoral Committee Members: Dr. Santosh Biswas, Dr. Arnab Sarkar, and Dr. Nagarjuna Nallam for their productive and constructive suggestions for my thesis work. I firmly believe that their opinions and comments help me to shape up my final thesis. Additionally, my sincere thanks to Prof. S V. Rao, the Head of the Department of Computer Science and Engineering, and other department faculty members for their constant support and helps. Furthermore, my sincere thanks to Prof. Kalpesh Kapoor for various academic/non-academic suggestions.

During my Ph.D., I got the opportunity to work with Dr. Shirshendu Das, Dr. Shounak Chakraborty, Khushboo Rani, Palash Das, Sheel Sindhu Manohar, Arijit Nath and Priya Sharma. For the countless number of times, sharing of knowledge and fruitful technical discussions with them helps me to carry out my research work. I especially thanks to my two seniors Dr. Shirshendu Das and Dr. Shounak Chakraborty for their initial guidance in the computer architecture simulators and the research. Various academic and non-academic discussions were always fun with them. I would also like to thank Dr. Sonal Yadav, Manik Bhosale, Rodney Stephen Rodrigues, Neelkamal, T. Susma Devi, Lt. Col. Alankar V. Umdekar, Sourabh Gavhale, Gibran Iqbal and Dipika Deb for helping and coordinating during the TA duties.

My friends and well-wishers have greatly helped me and without their help my work would not have been possible. I feel lucky to have a company of Dr. Shounak, Khushboo, Saptarshi, Hema, Dr. Bala, Manoj, Surajit, Deepak, Deepika, Nayantara, Arijit, Sheel, Palash (all PhDs). An uncountable number of random discussions with them makes my life not dull and joyful. My thanks are also due for Rohit, Saurav, Nitin, Ajay, Deepak, Vikas, Suryadeep, Ishank, Gopal, Sagar, Genius and many more (all College and School mates) for their tremendous support to me which hardly few people will get.

I sincerely thanks to Mr. Raktajit Pathak, Mr. Nanu Alan Kachari, Mr. Bhriguraj Borah, Mr. Monojit Bhattacharjee, Mrs. Gauri Deori and all other department staff members for helping me different ways and at different times during my stay at IIT Guwahati. I would also like to thanks the student affairs section for providing on-campus hostel facility. Last but not least, I am conveying my appreciation to security guards, janitors, hostel mess, and canteen staffs for making my life smooth in IITG campus.

I would also like to acknowledge the Tata Consultancy Service (TCS) for the financial grants provided under TCS RSP program. The travel grants under TCS scheme allows me to attend various national and international conferences that helps me to broaden my vision in the other areas of computer architecture. Furthermore, many thanks to Mr. Sunil Sarma from finance and account section and Mr. Sajal Mani Pathak from academic affairs for handling TCS related stuff.

Above all, I am incredibly fortunate to have moral support and encouragement from my parents and sisters: Shipra and Suchi. I must say without their moral support and continuous motivations, none of this would have been possible. Thank You for everything to me.

Last but not least, I would like to thank those people who discourage, demoralize, and menace me at different stages of life. All these motivate me to take something as a challenge and work hard for it.

Abstract

The ever increasing demand for higher processing speed with hiked data parallelism force the computer architects to increase the number of processing cores on a single chip called Chip-Multi-processors (CMPs). Towards meeting the performance goals, these CMPs are equipped with larger on-chip Last Level Caches (LLCs) to enhance the probability of the presence of data on-chip during process execution. The existing literature portrays that conventional LLCs built-in charge-based memory technologies are although faster but fall short in fulfilling these demands, especially in terms of increased power consumption. Furthermore, stagnation in process technology drove memory architects and researchers towards investigating Non-Volatile Memories (NVMs) for designing on-chip LLCs due to their promising scalability, reduced leakage power consumption, and compatibility with the conventional CMOS. However, many of these NVMs suffer from costly write operations with lower write endurance.

To achieve the best of both conventional as well as emerging NVMs, Hybrid Cache Architecture (HCA) has been evolved where different memory technologies are fabricated to build up a single level of cache. In particular, in this thesis, we adopt HCA based LLC, in which a large portion of LLC is built-in NVM for stimulating energy efficiency and the remaining smaller part is engineered with the conventional faster SRAM. In such an HCA, the block placement to the appropriate region is the key challenge from the energy-efficiency perspective. Towards this, we proposed a private block-based block placement technique that allocates dataless entries in the NVM portion of HCA. In this approach, additional savings in the number of writes to the NVM portion are governed by employing a Reuse Distance Aware Write Intensity Predictor. Besides the block placement approach, the fields of the predictor are used to improve the victim replacement decision for different portions of HCA. From this contribution, we get 34.5% reduction in writes and $16-19.6\%$ savings in energy over prior works. Towards a performance perspective, to overcome the effect of costly write latency operations, in the next contribution, the victim cache is explored with pure NVM and HCA based cache. With NVM cache, the victim cache is used to retain both victims as well as the write-intensive live blocks to save on the time to exchange and subsequent slow writes. By experimental analysis, we achieved 5.88% speedup over the baseline. With HCA, two policies are proposed to manage the victim cache effectively, where former one decides the placement of the block upon a victim cache hit in the different regions of HCA, whereas latter one gives a substantial amount of space

for the victims evicted from each region using dynamic region-based victim cache partitioning. These couple of approaches improve the overall performance of HCA by 4.43% and reduce the miss rate by 7.81%.

According to the available literature, due to lower write endurance, the lifetime of NVM caches is limited. Additionally, the run-time behavior of the applications, working set sizes, and cache replacement policies altogether lead to write variations across and inside the sets in the cache. In that, some sets and ways (inside the same set) get written heavily compared to others which are termed as inter-set, and intra-set write variations, respectively. These variations are one of the biggest design concern for HCA as they further limit the longevity and lifetime of its NVM-portion. To mitigate these unwanted write variation, two wear-leveling techniques: inter-set and intra-set are further proposed. The intra-set wear leveling works on the basic concept of the write restriction by partitioning the cache horizontally and vertically and is able to reduce the intra-set write variation in the range of $80-86.5\%$ with 7.27 times improvement in lifetime over the prior works. The inter-set wear-leveling technique exploits the concept of fellow sets and the dynamic associativity management to overcome the write variation across the cache set. With these approaches, write variation is reduced by $27.6-34\%$, and the lifetime is further improved by $14.7-20.7\%$ over the baseline.

The thesis has thus demonstrated the effective management techniques for longevity enhancement of the NVM cache for an optimal lifetime and controlling the effect of costly write operations.

Contents

भारतीय प्रौद्योगिकी संस्थान गुवाहाटी
Indian Institute of Technology Guwahati

List of Figures

List of Tables

भारतीय प्रौद्योगिकी संस्थान गुवाहाटी
Indian Institute of Technology Guwahati

Abbreviations

CMP	Chip Multi-Processor
NoC	Network on Chip
IPS	Instructions Per Second
IPC	Instruction Per Cycles
AMAT	Average Memory Access Time
CPI	Cycles Per Instruction
LLC	Last Level Cache
SRAM	Static Random Access Memory
DRAM	Dynamic Random Access Memory
NVM	Non-Volatile Memory
MRAM	Magnetic Random Access Memory
STT-RAM (or STT)	Spin Transfer Torque Random Access Memory
PCRAM (or PRAM)	Phase Change Random Access Memory
ReRAM (or RRAM)	Resistive Random Access Memory
HCA	Hybrid Cache Architecture
RDAWIP	Reuse Distance Aware Write Intensity Predictor
SWWR	Static Window Write Restriction
DWWR	Dynamic Window Write Restriction
DWAWR	Dynamic Way Aware Write Restriction
MWWR	Module Wise Write Restriction
RP	Reserve Part
NP	Normal Part
TGS	TaG Storage
FSSRP	Fellow Set with Static Reserve Part
FSDRP	Fellow Set with Dynamic Reserve Part
VC	Victim Cache
WLVC	Write Latency aware Victim Caching
AVBP	Access based Victim Block Placement
RDVCP	Region based Dynamic Victim Cache Partitioning

IRVP	Interval wise Region based Victim Placement
CMD	Cache MetaData
RDT	Reuse Distance Table
WBKI	Write-Back Per Kilo Instruction
ULC	Upper Level Cache
WRW	Write Restricted Window/Way
LRU	Least Recently Used
MRU	Most Recently Used
WLRU	Weighted LRU
MPKI	Miss Per Kilo (Thousand) Instructions
NUCA	Non Uniform Cache Access
UCA	Uniform Cache Access
SNUCA	Static NUCA
DAM	Dynamic Associativity Management
EDP	Energy Delay Product (Energy $\times$ CPI)

Chapter 1

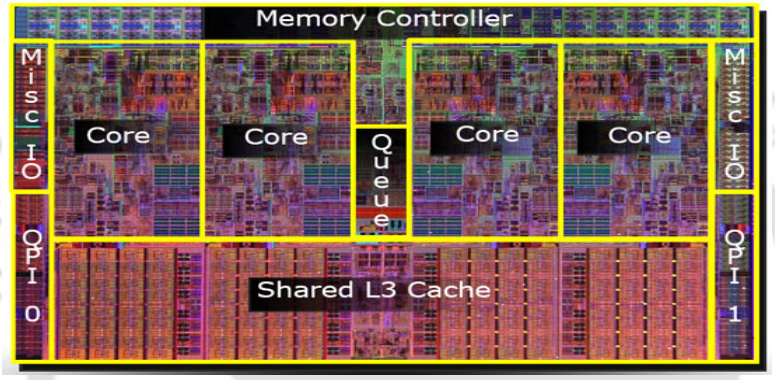
Introduction

Moore's law [1] gives us more and more transistors on the chip with every process generation. Earlier these additional transistors were used for improving the performance of processor architecture by adding more complex and simple pipelines and the better arithmetic and floating-point units. All these improvements enabled faster-performing processors due to better clock frequencies. However, if the clock frequency is increased at the same rate, it would be challenging to manage the heat dissipated by the processor. Hence instead of improving on the frequency, researchers moved towards the multiple cores on the chip to keep up with Moore's law. We are now in an era of multi-core and many-core processors on the chip, popularly termed as CMPs.

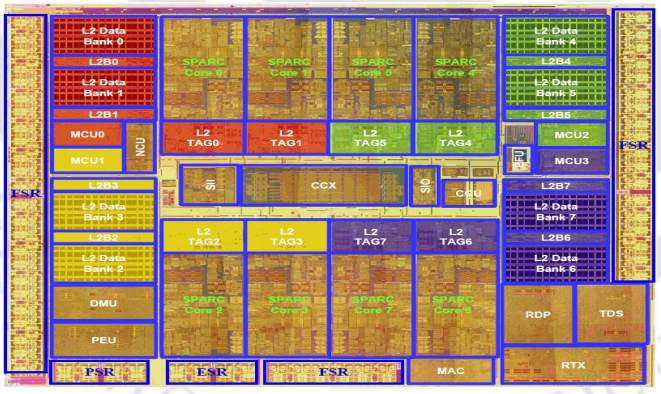
1.1 Modern Chip Multi-Processors (CMPs)

In CMPs, the large number of cores are attached on the same die, with identical (homogeneous) or different (heterogeneous) power budgets. The cores employed in the CMP are either complex or Simple super-scalar processors that can perform a variety of tasks. The CMPs are initially designed to improve the performance and to add the parallelism. In addition, the benefit of employing the CMPs is that the performance can further be enhanced by integrating more number of

identical cores in each new generation. As in the future, with each upcoming generation, the workloads or applications need higher throughput and parallelism with the larger data demands. To fulfill the data demand, the multi-level on-chip caches are attached. Whereas to achieve parallelism, the CMPs becomes larger and it needs a strong communication infrastructure which can only be provided by the modern communication system, called Network-on-Chip (NoC). Figure 1.1 presents the design and the floor-plan outlines of two modern CMPs architectures: Nehlam and UltraSparc T2 [2].



(a) Nehalem [3]



(b) UltraSPARC T2 [4]

FIGURE 1.1: Modern CMPs: design and floor-plan outlines

The modern CMPs are built by using the following components:

1. The processing cores as the computation unit, CPU cores.
2. The communication medium through which on-chip communication between different component happens, Network-on-Chip.

3. An on-chip storage to retain most of the required data needed by the running application, On-chip Caches.

The computation performance of the CMPs are measured in terms of Instruction Per Second (IPS), speed-up factor, Total Execution Time, Instruction Per Cycle (IPC) to a base system etc [5]. In particular, the computation performance depends upon the time taken to execute the application and the parallelism achieved by the applications while execution. The parallelism of the application is achieved by using the concept of multi-threading, where the application is spawned into multiple threads.

Based on the nature of the instruction, the application execution on the CMPs is categorized into two categories: Memory Bound and Compute Bound [6, 7, 8, 9]. In compute-bound applications, the majority of the instructions are intended for the arithmetic and logical operations. Whereas, with memory-bound applications, the majority of the instructions are intended for memory operations, e.g., load and store. Therefore, with this kind of applications, the system performance has directly relied on the two factors: memory access latency and the core clock frequency. In the above-mentioned dependencies, the memory access time further relies on the following factors: (1) Time taken by the memory to reach the block that has been requested (2) Time taken by memory system to deliver the block to requestor core. The former one depends on the type of memories employed in the hierarchy. Whereas, the latter one relies on the performance of communication infrastructure, i.e., Network-on-Chip. Thus, based on the above discussion, the total execution time of the system (by assuming simplistic model) can be modeled as:

$$Total\_Exec\_Time = Computation\_Time + Memory\_Cycles + NoC\_Latency \tag{1.1}$$

1.2 Cache Memory

One of the essential part of the memory hierarchy to improve the performance of the system is the Cache Memory. It is the smallest and the fastest storage that contains most of the required data needed by the applications running on CMPs. Thereby, it restricts the block accesses that directly go to the main memory every time and thus reduces the average memory access time. In other words, for each memory access, the processor core first requests to the cache. If the requested block is present in the cache, it is directly supplied to the core. Otherwise, it is fetched from the main memory. Bringing the block from the main memory takes extra clock cycles than accessing the cache. However, the two principles: temporal and spatial locality of the cache guarantees that once the block is placed in the cache, it will be used multiple times (more likely) before getting evicted. Thus, better the hit rate of the cache memory, lesser is the time consumed by main memory accesses. By this way, the cache memory improves the performance of the system by limiting the Average Memory Access Time (AMAT). Lesser the AMAT, lower is the Cycles Per Instruction (CPI) or higher is the Instruction Per Cycle (IPC).

With the rapid advancement in the CMOS in each process generation, the reduced channel length of modern transistors [10] facilitates the computer architects to employ large number of large-sized, multi-level on-chip caches. These large sized multi-level caches are labeled based upon their distance from the processing core. In particular, the level which is nearest to the processing core is labeled as the L1 cache (divided into instruction and data cache). The next level to L1 is labeled as L2 (which is a unified cache) and so on [5]. Among these multi-levels, the on-chip Last Level Cache (LLC) plays the most prominent role as it contains most of the required data to save from the costly main memory accesses. The LLCs are larger in size with a significant area overhead on the wafer real-estate (as it can be seen from the figure 1.1). Most of the recent CMPs generally have two to three levels of on-chip caches. Based on this, most of the work presented in this thesis is limited

up to the three-level, set-associative cache. The set-associative cache is considered as the best architecture compared to direct-mapped and fully associative cache.

1.3 Cache Memory Architecture in CMPs

For CMPs, accessing a larger cache is both power and time-consuming. Besides, the large caches are also accounted as a significant contributor to on-chip power consumption. To solve this, cache architects divide the larger cache into multiple small slices called a cache bank. This division is based on the two-levels of patterns: (1) Way based division in which the way(s) from all the sets are made the cache bank (2) Set-based division in which set(s) from all the ways made the cache bank. These multi-bank small cache structure not only reduce the access latency but also provides more design space to the circuit designers to fabricate different memory technologies for better power optimizations.

The CMP cache architectures are mainly of two types [11] based on the physical placement of the LLC: (1) CMP with shared LLC (2) CMP with private LLC. Whereas, in both the caches architectures, the L1 instruction, and data cache are private to a core. In the former LLC placement, the LLCs are larger and shared to each core. The data block is present only at one location of the cache, and all the processing cores share only this copy. In shared LLC, based on workloads executed by the processing core, the cache storage is dynamically allocated to the core. On the other hand, in the latter LLC placement, the LLCs are small and private to a core. The private LLC is present near to the core and thus provides faster cache accesses compared to shared LLC. Based on the requirement by the cores, one data block may be present in multiple private LLCs. Thus, the private LLC size becomes limited, and it experiences more capacity misses. Considering to both type of placements based on the growing size of workloads, the shared LLC will be a better design choice. Thus, in this dissertation, we have considered shared LLC in all of our works.

The rest of the dissertation considers the LLC as the shared LLC.

1.4 Power Consumption in CMPs

For each of the component present in the CMP, the total power consumption can be divided into three categories [12, 13, 14]: (1) Static Power, (2) Dynamic Power, and (3) Short Circuit Power.

1.4.1 Static Power

The static power is defined as the power drawn by the on-chip circuitry, even when the circuitry is not performing any task. Static power resembles the leakage power of the circuit, which is based on two important leakage units: (1) Gate Leakage, and (2) Sub-threshold leakage [14, 15, 16, 17]. Between these two units, the sub-threshold leakage has a direct relation with the chip temperature and the supply voltage. As the chip temperature becomes higher, the covalent bond of the atoms in the semiconductor material is broken that releases the electrons which flows in the reverse bias and generates the current, called the sub-threshold leakage current. The power drawn due to the sub-threshold current is called sub-threshold leakage power. Whereas, the gate leakage power is due to the down-scaling of device size and the reduction in the thickness of gate oxide material. The direct relation of running chip temperature and supply voltage on the sub-threshold leakage is presented in the following equation [18, 15]:

$$P_{stat} = K_1 V_{DD} T^2 e^{(\lambda V_{DD} + \beta)/T))} + K_2 e^{(\gamma V_{DD} + \delta)} \tag{1.2}$$

In the eq. (1.2), P_{stat} represents the static power consumption due to sub-threshold leakage by a CMOS circuit. T denotes the current temperature and V_{DD} implies the supply voltage. K_1, K_2, λ, β, γ, and δ are the empirical constant that represent the different circuits parameters.

1.4.2 Dynamic Power

The dynamic power is defined as the power consumed due to on-chip circuity while performing the task. It is due to the switching activity of the transistor while charging/discharging the output capacitances. The dynamic power is represented by the help of the following equation [14, 15]:

$$P_{Dyn} = \alpha.C.V^2.f \tag{1.3}$$

In the eq. (1.3), P_{Dyn} denotes dynamic power of cores. The parameters α, C, V and f represent the activity factor, capacitance, supply voltage and frequency of the core, respectively.

For the on-chip caches, the dynamic power is consumed during the cache access. The cache is accessed either for the write operation or for the read operation. In the traditional cache (fabricated from the SRAM/DRAM), the read and write access power consumption are same [13, 19]. Whereas, with the advancement of the semiconductor technologies, researchers have also considers the emerging Non-Volatile Memory (NVM) technologies for the caches. In these NVM caches, the access power for the read and write operation is asymmetric [20, 21, 22]. The detailed modeling of the power consumption for different types of memory technologies used for the fabrication of the cache is discussed in Chapter 2.

Whereas, for the NoC, the major chunk of dynamic power is consumed by two basic units: (1) Routers, and (2) Connecting Links. Among these two units, the routers are the most complex part of the system that uses multiple routing algorithms for sending the data optimally across the chip. The data is passed through the connecting link, which is nothing but a set of metallic wires. During the routing operation, the dynamic power consumed by the router is due to the following three units [23]:

1. **Router Clock:** The essential part of the router that maintains the synchronization.

2. **FIFO Buffers:** The buffer maintains the sequence/order of incoming/outgoing data blocks in the router.

3. **Arbiters and Allocators:** Make sure that the data block reaches to the proper destinations.

1.4.3 Short-Circuit Power

The short circuit power is defined as the power consumed due to non-zero fall/rise time of the CMOS circuitry. In particular, it is the power consumed during the short-time span when both NMOS and PMOS are active simultaneously. The short circuit power consumption is very negligible and most of the time ignored during the calculation of power consumption of CMPs [14, 15, 16].

This dissertation considers only the static and dynamic power/energy for the calculation of power/energy consumption.

For the modern CMPs, among the different on-chip components, the processing cores are usually accounted for high dynamic power consumption. Whereas, the on-chip caches fabricated from the SRAM/DRAM are considered for their high leakage energy. Also, in the modern CMPs, it has been noticed that the as the size SRAM/DRAM cache become larger, the number of transistor increases and occupies large wafer real estate area. This leads to large leakage power consumption which becomes the significant contributor in the total power consumption of the chip. Table 1.1 presents the percentage power contribution by the on-chip SRAM caches with respect to the total power consumed by the chip [12, 24]. These large numbers in the table motivate the computer architects to reduce the power consumption by the on-chip caches. This thesis focuses upon reducing the high leakage power consumption of the on-chip SRAM caches by employing the emerging near-zero leakage power NVM caches.

Microprocessor	Power Consumed by on-chip Caches with respect to total power
ARM 920T	44%
Strong ARM SA-110	27%
21164 DEC Alpha	25-30%
Niagra	12%
Niagra2	21%
Alpha 21364	13%
Xeon (Tulsa)	13%

TABLE 1.1: Power consumed by on-chip caches

1.5 Memory Technologies used for Caches

Lastly, for around three decades, the cache hierarchy employed in the computer design has been essentially the traditional charge based SRAM/DRAM memory technologies. Their advantageous properties are low access latency, very high write endurance, efficient dynamic energy, and manufacturability. However, they also consume high leakage energy and occupy a large wafer real estate with each process scaling. The previous studies [12, 24] on these traditional memory technologies illustrates that, in the existing microprocessors, the power budget consume by the on-chip cache ranges from 12-44%. In addition to this, one study [25] pointed out that to limit the memory bandwidth, large-sized SRAM caches are to be used that may occupy 90% of the chip for future CMOS generation. To counter these challenges, different systems and architectural efforts have been made over the previous years [24]. But the performance target needed by the modern processors may not be fulfilled by the traditional SRAM based caches.

Recently, because of the several desired features like non-volatility, higher density and lower leakage than SRAM, the emerging Non-Volatile Memory technologies have been considered as a choice for the memory hierarchy by the computer architects [20, 26]. These NVM technologies include Phase-Change RAM (PCRAM), Magnetic RAM (MRAM) and Resistive RAM (RRAM), etc. Unlike traditional memory technologies that use charge as an information carrier to store the bit value, the NVM technologies use resistance as an information carrier. Besides this, the NVM technologies can store multi-bit information in the cell that increases the cache capacity within the same area footprint. Even though, with ongoing research work, some of the NVMs are at an early stage of development

and some of them have reached the commercial product stage. However, despite many advantages offered by these NVMs, they still lack behind as the alternate memory choice compared to traditional memory technologies. This is due to costly write operations and the weak write endurance of these technologies. Thus, utilizing these emerging NVM technologies at the levels of cache hierarchy is a major problem/challenge for the computer architects.

More details about the working methodology and the preliminary concepts and characteristics of different memory technologies is discussed in Chapter 2.

1.6 Motivations

From the energy perspective, the cache fabricated from the NVM technologies saves a lot of leakage power compared to the traditional SRAM/DRAM based caches. In particular, for conventional caches, the leakage power is the main contributor in the total power consumption. Whereas, comparatively for the NVM memories, due to near-zero leakage power, the total power consumption is less with process scaling and with the increase in the size of the cache. However, the NVM memory suffers from costly write latency and energy, due to long time and large energy consumption for the bit flipping. To mitigate the expensive write operations, researchers make use of the best characteristics of each memory technologies by using Hybrid Cache Architecture (HCA) [27, 28]. In HCA, the cache is partitioned into multiple regions made up of different memory technologies. In HCA, the placement of the appropriate block in the proper region is always a major concern/problem for energy efficiency. On the other hand, from the view of performance, the performance reduction due to the costly write operations in the NVM cache and the increase in the miss rate by partitioning the hybrid cache generates another set of problems. All these phenomena motivate us to develop a block placement policy for the hybrid cache and as well as develop strategies to improve the performance of hybrid and non-volatile cache by using victim cache.

Another leading set of the problems with the employment of NVM caches arise due to weak write endurance. Compared to the traditional memory technologies where the write endurance is more than 10^{15} writes, different NVMs have a write endurance in the range of 10^5 to 10^{12} writes [20]. With these write endurance values, the NVM cache will last only for certain minutes or days. Furthermore, it has been observed that the write accesses to the LLC is non-uniformly distributed among the cache sets, i.e. inter-set and among the blocks inside the cache set, i.e. intra-set. In particular there is a variable write distribution between the cache set where some of the cache sets are heavily written while some others are lightly written. The same case is applicable to the blocks inside the cache set. In other words, some of the blocks inside the cache set are heavily written while some are lightly written. This non-uniform dispersion of the writes inside the cache creates certain write hotspots at different levels of cache, i.e., inter-set and the intra-set [29]. These write hotspots along with the weak write endurance affects the lifetime of the non-volatile cache as a whole. All these circumstances encourage us to develop strategies to maintain the uniformity in the writes and improve the lifetime.

The main aim of the research work is to enhance the longevity of the non-volatile cache and utilize them as a better candidates for the last level caches. To facilitate this process, we make different contributions in the following directions:

- A private block and reuse distance aware write intensity prediction based block placement technique to save the costly write operations in the hybrid cache.
- Write restriction based approaches to mitigate the intra-set write non-uniformity.
- Fellow set based dynamic associativity management strategies to overcome the inter-set write non-uniformity.
- Improving the performance of NVM and Hybrid cache by associating a victim cache with them.

1.7 Thesis Contributions

The major contributions of this thesis can be summarized as follows:

1.7.1 Private Block and Prediction based Block Placement approach in HCA

In this approach, we have presented a block placement technique that places the different categories of blocks to the different regions of the hybrid cache. The proposed idea is motivated by the facts that there are a considerable amount of private blocks present in the LLC, and most of the time, these blocks contain worthless and stale data. For such data entries, the actual worthy data is present in the LLC at the time of first write-back operation from the upper-level cache. We have identified all these facts, and our block placement approach is built on the top of these concepts. In our block placement approach, when the private block is loaded into the LLC on a miss, a dataless entry is allocated into the non-volatile region of hybrid LLC. For such dataless entries, the appropriate tag entry is made during the allocation of the block in the cache. In such dataless entries, the actual worthy data is made at the time of first write-back operation. Whereas, for the blocks (called as a shared block, including instruction block) other than the private blocks, the normal block fill operation is performed to the NVM region when such block is loaded in LLC on a miss. Eventually, the block is migrated to the SRAM region, when the second write back is performed on the entry in the LLC. Besides the private block-based placement, to further save the writes in the NVM region, we have employed a Reuse Distance Aware Write Intensity Predictor (RDAWIP). RDAWIP predicts the placement of the first write back operation for the dataless entries. In addition to this, we have proposed a replacement policy that works on top of the traditional replacement policy. The replacement decision is based on the different fields of the predictor and the reuse distance of the block. The results of different variations of the proposal are compared with two existing techniques:

Read Write Aware Hybrid Cache Architecture (RWHCA) [28] and Write Intensity (WI) [30]. In particular, the following variations are proposed:

- **P:** Policy with dataless entry, i.e. policy with private block based data placement.
- **T:** Policy with private block based data placement along with the prediction of the first write operation through RDAWIP.
- **M:** Policy with private block based data placement along with the augmented replacement policy that decides with the help of RDAWIP.
- **N:** Policy with private block based data placement along with the prediction of first write-back and replacement policy.
- **S:** Policy with private block based data placement that makes the prediction and replacement decision based on the sampler.

The different variation of the policies saves the writes in the non-volatile region in the range of 41.9% - 48.1% (17.5% - 25%) over RWHCA and 27.4% - 35.1% (7% - 15.3%) over WI for Dual (Quad) core system. Whereas, the energy gains by the different variations are in the range of 29.2% - 34.3% (16.1% - 19.6%) over RWHCA and 17.3% - 23.3% (10.3% - 14.1%) over WI for Dual (Quad) core respectively. The proposed techniques maintain the same performance over the baselines.

This work is fully discussed in Chapter 3.

1.7.2 Intra-Set Write Variation mitigation using Write Restriction

In this work, four approaches: SWWR, DWWR, DWAWR and MWWR are presented using different partitioning techniques and at different levels of a cache bank. These levels are classified as follows:

- DWAWR: At the level of cache-ways.
- SWWR, DWWR: At the level of window by partitioning the cache vertically into multiple equal-sized windows (in such a way that each window contains an equal number of cache ways).
- MWWR: At the level of module by partitioning the cache horizontally into multiple equal-sized modules (in such a way that each module contains an equal number of cache sets).

All these intra-set wear leveling approaches uses the basic concept of write restriction to overcome the write variation or non-uniformity. In the write restriction, during the application execution, for the certain predefined interval, different window/ sub-ways (inside the module)/ cache ways are treated as the write restricted or read-only. The selection of window/sub-ways/ways for the write restriction is based on the following methods: (1) Using round-robin method, and (2) with the help of write counters associated with each window/way/sub-way. The results of the proposed approaches are compared with four existing intra-set wear leveling approaches: Polf [29], EqualChance [31], Write Aware Displacement [32] and WriteSmoothing [33].

This work reduces the write non-uniformity in the range of 80% - 86.5% with negligible degradation in the performance and outperforms over all the prior works. The lifetime gains by the proposed approaches are in the range of 4.8 - 7.27 times.

The detail description of this work is given in Chapters 4 and 5

1.7.3 Inter-Set Write Variation mitigation using Dynamic Associativity Management

In this work, to mitigate the non-uniformity in the write distribution between the cache sets, two strategies: FSSRP and FSDRP are proposed. Both the proposals are based upon fellow sets and uses the concept of Dynamic Associativity Management. In these approaches, the cache sets are logically grouped into different

fellow sets. Each cache set is divided into two logical parts: Normal and Reserve Part. During the application execution, the normal part of the heavily written fellow sets spread out their writes in the reserve part of the lightly written fellow sets of the fellow group. Based on the position of the reserve part of the fellow sets, two variations of the approaches is presented. In the first approach, the position of the reserve part is fixed and static. Whereas in the second approach, the reserve part position is dynamic and it keeps moving over the cache after a certain period of execution. The results of the proposed works are evaluated with one prior work: Swap Shift [29] and the baseline for quad-core system.

The strategies reduce the non-uniformity in the writes between the cache sets in the range of 27.6% - 34% with negligible degradation in the performance. The lifetime gains by the proposed approaches are in the range of 14.7% - 20.7%. We have also seen the significant improvement in the different result metrics over the prior works.

A more detailed description of this work is given in Chapter 6.

1.7.4 Victim Caching to Improve the Performance of NVM Caches

To fill the performance gap generated due to costly write operation in the NVM cache, we have integrated a victim cache with the main cache. Employment of the victim cache with the NVM cache requires good victim migration and retention capabilities. As the migration of the block from the victim cache to the NVM based main cache requires extra clock cycles as well as incurs extra energy with the NVM caches. By considering these facts, we have developed a strategy that serves write-intensive blocks directly from the victim cache without placing them back to the main cache. In addition to this approach, we have also proposed a replacement policy for the victim cache. The proposed replacement policy has taken the victim replacement decision based upon the number and the type of requests entertained by the block in the victim cache as well as the time-stamp at

which the block is last accessed. The result of the proposed scheme is compared with the existing hybrid cache architecture: RWHCA [28] and the baseline SRAM and STT-RAM-based caches for the quad-core system.

The proposed technique improves performance by 5.88% over STT-RAM and 3.45% over RWHCA. Whereas, the energy improvement values over baselines STT and SRAM and the prior works: RWHCA are 8%, 93.5%, and 78.85% respectively. All the improvements come at the marginal cost of storage and area overhead.

The full description of this work is given in Chapter 7.

1.7.5 Victim Caching to Improve the Performance of Hybrid Caches

In this proposal, we have added a victim cache to compensate the performance gap due to increased miss rate by partitioning the cache into multiple regions, and the applied block placement policy. The employment of the victim cache with the hybrid cache requires an effective victim block placement policy upon hit in the victim cache to the different region of the hybrid cache and to give a substantial amount of space for the victims evicted from each region of hybrid cache in the victim cache, a region-based dynamic victim partitioning policy. All these facts motivate us to develop an access based victim block placement policy and the dynamic region-based victim cache partitioning strategy. Our access based victim block placement policy has considered the type of access as well as the dirty status of the victim block before making a placement decision to the different region of the hybrid cache. Whereas, our dynamic region-based victim partitioning approach dynamically partitions the victim cache based upon the number of victim evictions from the smaller region of the hybrid cache. The victim eviction counts are assimilated by dividing the application execution into multiple intervals. The results of the proposed scheme are compared with baselines SRAM, STT-RAM and the hybrid cache with no specific placement policy and prior hybrid cache architecture: RWHCA [28] on a quad-core system.

The proposed work improves the performance of hybrid cache by 4.43% against STT, 3.03% against baseline HCA, and 2.32% over RWHCA. Whereas, the respective improvements in the energies are 41.3% against SRAM, 34.1% over STT, 24.3% against HCA, and 15% over RWHCA. All these improvements come at a marginal cost of storage and area overhead.

More details about this work are given in Chapter 7.

1.8 Summary

To fulfill the high data demands in the modern CMPs, integrated with a large number of processing cores, larger multi-level on-chip caches are attached. Among these multi-level caches, the LLCs play an essential role in maintaining system performance. But these larger sized LLC fabricated from traditional memory technologies occupies larger wafer real estate area and also accounts for significant leakage power consumption. The recent emergence of Non-Volatile Memory technologies has shifted the paradigm and the computer architects are looking at them as a alternate choice in the memory hierarchy. Over the traditional memory technologies, these NVMs allow the construction of the on-chip LLC which are highly dense, non-volatile, low static energy, and better scalability. However, when employed in the cache, these NVMs incur extra write energy and consume extra clock cycles for the write operations. In addition, these NVM caches also have a minimal lifetime due to the weak write endurance and the non-uniform write distribution (at the level of cache-sets, i.e., inter-set and blocks inside the cache sets, i.e., intra-set) from the higher-level caches. In this dissertation, we aim to enhance the longevity of the NVM based LLC by dealing with their challenges and make them as a capable candidate to fit in this cache hierarchy.

To overcome the costly write operations, we initially employed a hybrid cache architecture where a larger portion of NVM, a small portion of SRAM is integrated to save the costly write operation. In such an HCA, block placement is a critical task for energy efficiency. In this regard, we have presented a block placement

technique that considers the private blocks from the different memory block access and uses a predictor to effectively place the different blocks in the different regions of HCA. Whereas, with regards to compensating the performance gap due to costly write operations for the NVM cache and the increased miss rate for HCA, we have employed the victim cache with both the architectures. In this work, with NVM cache, we have presented selective victim retention and caching policy for the write-intensive blocks. Whereas, with HCA, we have proposed an access aware block placement technique for placing the block from the victim cache upon a hit and to give substantial space for the victim evicted from each region of HCA, a dynamic region-based victim cache partition approach is presented.

To improve the lifetime of the NVM caches affected by the non-uniform write distributions, we propose two wear-leveling approaches for inter-set and intra-set. For intra-set wear leveling, four approaches: SWWR, DWWR, DWAWR, and MWWR are presented that works on the basic concept of the write-restriction. Whereas, for inter-set wear leveling, two strategies: FSSRP and FSDRP are proposed that works on the basic idea of fellow-set and dynamic associativity management. Both the wear leveling proposals improve the lifetime significantly with the negligible impact on performance.

1.9 Organization of Thesis

The rest of this thesis is organized as follows:

- Chapter 2 summarizes the background and prior works related to the contributions of the thesis.
- Chapter 3 presents the first contribution, which is the private block and prediction based block placement and the replacement technique for the hybrid cache.
- Chapters 4 and 5 illustrate the four intra-set wear-leveling policies to improve the lifetime of non-volatile cache.

- Chapter 6 discusses two inter-set wear-leveling strategies to enhance the lifetime of non-volatile cache.
- Chapter 7 explore the use of the victim cache with the NVM/HCA based main cache to improve the performance affected due to the costly writes and miss rate.
- Chapter 8 finally concludes the thesis.
- Appendix A summarizes the simulation framework used to build up the contributions of this thesis.

भारतीय प्रौद्योगिकी संस्थान गुवाहाटी
Indian Institute of Technology Guwahati

Chapter 2

Background

As presented in Chapter 1, the conventional SRAM/DRAM based LLC in modern CMPs are usually shared and larger in size. With regards to the next generation workloads, apart from the alarming rise of leakage power consumption with the process scaling, the traditional charge based LLCs also fall behind in terms of scalability and the lower density. The main goal of this thesis is to utilize the emerging NVM technologies as the Last Level Cache by considering their weak write endurance and the costly write operations as their main constraints. We initially summarize the preliminary concept of current memory system and the emerging NVMs. Also, we have discussed the challenges with the employment of NVMs in the cache hierarchy. Later we describe the state-of-art energy/performance efficient techniques and the wear leveling approaches for the NVM caches developed over the years.

2.1 Cache Memory Technologies

The characteristics of an ideal memory technology are: fast, highly dense, reliable, low energy consuming and cheap. However, none of the single memory technologies fulfill all of these characteristics. For instance, the SRAM memories are fast, but at the same time, they are expensive, power hungry and have high feature size.

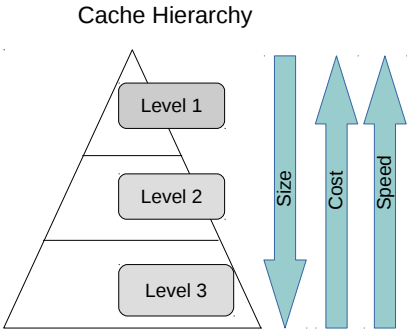


FIGURE 2.1: Characteristics required at each level of cache

DRAM memory technology is cheaper and has good density but it is unreliable and slower. The solid state flash memory is reliable and has high density then DRAM, but its write endurance, write energy, and latency are major concerns.

Fortunately, it is possible to design the different levels of cache hierarchy using different memory technologies that are fit for that level. This enables us to allow the formation of the system that gets the performance benefit from the use of faster technology, energy benefit from the use of power efficient components and cost advantage by using the cheapest level technology for the hierarchy. All these advantages are feasible due to the concept called locality of reference. Figure 2.1 present the characteristics required for each level.

2.1.1 Conventional Charge Based Memory Technology

Over the past three decades, different levels of the cache hierarchies have been fabricated from SRAM memory technology. Some of the prior studies [34, 35, 36] also exploits DRAM memory technology for the last levels of the cache hierarchy. Detailed explanation of these conventional charge based memories is given below.

2.1.1.1 Static Random Access Memory (SRAM)

The most familiar and prominent SRAM cell design used for the caches require six transistors. Figure 2.2 shows the schematic view of the SRAM cell. As can be seen from the figure, the core of the SRAM cell contains four transistors T_1 to T_4

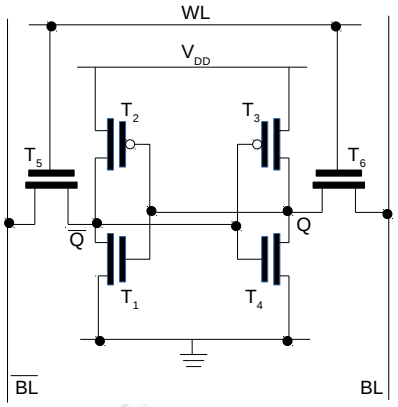


FIGURE 2.2: Schematic view of SRAM cell

that model two cross-coupled inverters and are used to store the bit information. The stored bit information in these transistors is represented in the form of two stable states, **0** and **1**. These saved states are permanent until the power (Vdd) is applied. The two additional transistors T_5 and T_6 are used for accessing the storage cell during the read and the write operation. The read and write operation for the SRAM cell are described below:

Read Operation: To access the state of a cell, the word access line WL is enabled. This makes the stored stable states available for the read operation in the lines BL and $\overline{BL}$.

Write Operation: To write the cell, the BL and $\overline{BL}$ are first set to the desired value, and afterward, the line WL is raised.

Other than the design perspectives, there are some other properties (that includes pros and cons) in the SRAM which are important to discuss:

1. The access speed of the SRAM cell is very fast. In particular, as soon as the WL is enabled, the stored stable state is available for the access.

2. The most common SRAM design requires six transistors, thereby it incurs more area on the wafer real-estate and has lower density than the other memory technologies.

3. The SRAM cell requires constant voltage supply to retain the data.

4. The SRAM cell is costlier in terms of cost/bit comparison.

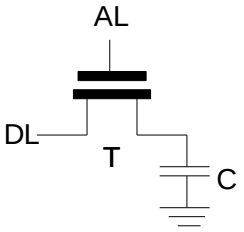


FIGURE 2.3: Representational view of DRAM cell

2.1.1.2 Dynamic Random Access Memory (DRAM)

Figure 2.3 shows the representational view of the DRAM cell. The DRAM cell is made up of a transistor and a capacitor. The state of the DRAM cell is stored in the capacitor C as a charge. To access the state, the transistor T is used. The read and the write operation for the DRAM cell are described below:

Read Operation: To access the state of the DRAM cell, voltage is applied to the access line AL. This makes either the current to flow on the data line DL, based upon the charge stored in the capacitor. In case, if there is no charge then no current flows to the DL.

Write Operation: To write the cell, the DL is appropriately set to the desired value. Afterward, the voltage is applied to AL for the extended period either to charge or drain the capacitor.

Because of the more straightforward structure of the DRAM cell, the feature size and the cost of the DRAM is lesser than the SRAM. Despite these advantages, the capacity of a capacitor to retain the charge in the DRAM cell is shallow, and it requires continuous refresh operation to hold the data correctly. Each refresh operation requires extra time and energy.

The other properties of DRAM can be summarized as follows:

1. The cell structure of DRAM is simpler, and thus, it allows the high density with low cost/bit comparison.

2. The access speed in the DRAM cell is slower compared to SRAM.

3. The refresh operations required to retain the data correctly makes DRAM power hungry.

2.1.2 Emerging Non-Volatile Memory Technology

Nowadays, there are several emerging Non-Volatile Memories (NVM) that are under research. One of state of art [37] lists 12 such NVM technologies: Spin Transfer Torque Random Access Memory (STT-RAM), Phase Change Random Access Memory (PCRAM), Resistive Random Access Memory (ReRAM), Ferroelectric Random Access Memory (FeRAM), Nano Random Access Memory (NRAM), Conductive-Bridging Random Access Memory (CBRAM), Single Electronic Memory (SEM), Polymer, Molecular, Racetrack, Holographic and Probe. From this list, some of the memories have reached advanced development stages and are commercially manufactured and some are not yet mature.

In this dissertation, we limit our study to the STT-RAM, PCRAM, and ReRAM as it is extensively studied and considered as a viable choice in the cache hierarchy. They are also backed by commercial industries.

2.1.2.1 Spin Transfer Torque Random Access Memory (STT-RAM)

The STT-RAM [38] is the new generation of Magnetoresistive Random Access Memory technology. Figure 2.4 (a) shows the conceptual view of STT-RAM cell [39]. The STT-RAM cell consists of Magnetic Tunnel Junction (MTJ) connected in series with access transistor. The MTJ contains two ferromagnetic layers separated by a thin insulating oxide tunnel barrier made up of MgO. One of the ferromagnetic layers, called the free layer, changes its magnetization direction by using spin-polarized current. While, the other layer, called the reference layer, keeps its direction fixed. The magnetization directions of these two layers are used to represent the data bit stored in the cell. When the magnetization direction of the free layer and reference layer are the same direction, the MTJ has the low resistance and represents '0' state of STT-RAM cell (figure 2.4 (c)). On the other hand, if the magnetization direction is anti-parallel, the MTJ has high resistance and represents '1' state of STT-RAM cell (figure 2.4 (c)). The read and write operation for the STT-RAM cell are described below:

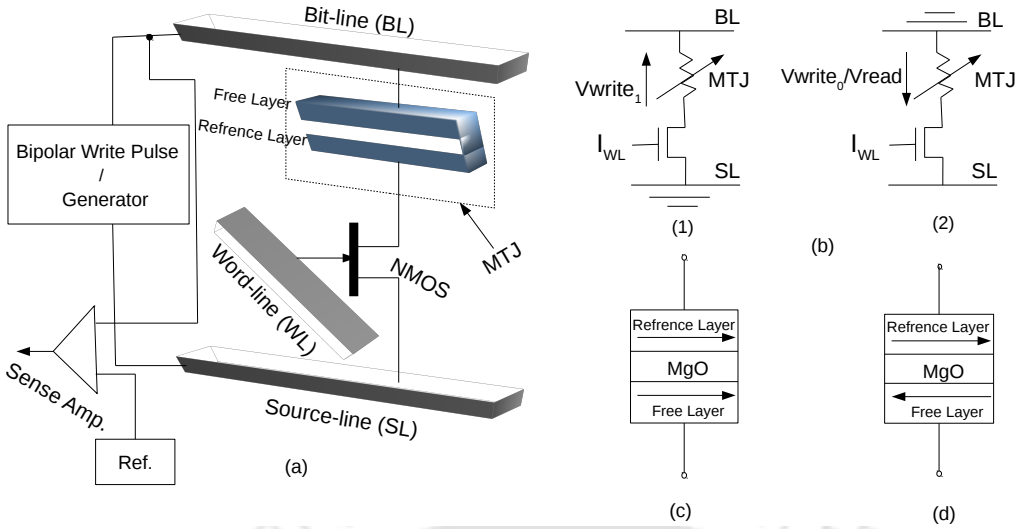


FIGURE 2.4: (a). Conceptual view of STT-RAM cell (b) Schematic STT view with (1) Write '1' operation (2) Write '0' and Read operation (c) Parallel low resistance, representing '0' state (d) Anti-parallel high resistance, representing '1' state

Read Operation: To access the state of the STT cell, the access transistor of the cell is enabled and the small voltage difference is established between the lines (source and bit line) (figure 2.4 (b) (2)). This effect causes the current to be generated across the memory cell, which is compared with the reference current through the sense amplifier.

Write Operation: To write bit '0', a large positive voltage difference is established between the source and bit line (figure 2.4 (b) (2)). To write bit '1', a large negative voltage is established between the lines (figure 2.4 (b) (1)).

The other important characteristics of STT-RAM are summarized below:

1. STT-RAM tries to attain better scalability by using different write mechanism based on spin polarization [40].

2. The STT-RAM cells have more density compared to SRAM, but have lesser density compared to DRAM [41].

3. Compared to PCRAM and ReRAM, the endurance of STT-RAM is excellent. However, when employed in the cache the endurance is still consider to be less [20].

4. The write operation of STT-RAM are costlier with respect to SRAM and DRAM [41].

Other than the design characteristics, the STT-RAM chip is commercially manufactured and available in the market. For instances, 4Gbit STT-RAM based perpendicular MTJ at 90nm technology node is fabricated by Toshiba and SkHynix incorporation [42]. Qualcomm and TDK-Headway built the 1Mbit STT at 40nm technology node [43]. Recently, Intel and Samsung fabricated 7.2M and 8M bit STT at 22 and 28 nm technology nodes [44, 45] respectively.

In this thesis, the terms STT-RAM or STT and PCRAM or PRAM are used interchangeably.

2.1.2.2 Phase Change Random Access Memory (PCRAM)

Currently the most mature emerging NVM technology that is under research is the Phase Change Random Access Memory (PCRAM) [46]. Figure 2.5 shows the representational view of PCRAM cell. The PCRAM cell contains phase change or chalcogenide material and an access transistor. The chalcogenide material is generally made up of GST ($Ge_2Sb_2Te_5$, or Germanium, Antimony and Tellurium) and shows two different phases: Amorphous and Crystalline by the application of heat. The high electrical resistivity characterizes amorphous phase and represents the RESET state of the cell. On the other side, the crystalline phase is characterized by low electrical resistivity and represents the SET state of the cell. The read and write operation of the PCRAM cell are described below:

Read Operation: To access the state of cell, a small voltage is applied across GST. This effect causes the current to be generated as there is a wide resistance gap exist between the amorphous and crystalline phase. The state of the cell is identified by sensing the pass through current with the help of access transistor and the word-line controlling.

Write Operation: To SET the PCRAM cell, a long duration moderate power

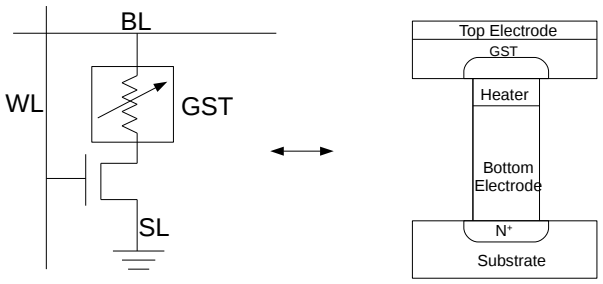


FIGURE 2.5: Representational view of PCRAM cell

pulse is applied that heats the GST above the crystalline temperature and makes the chalcogenide material crystalline. On the other side, to RESET the PCRAM cell, a high power pulse is applied that heats the GST above the melting temperature and makes the chalcogenide material amorphous.
The other important characteristics of PCRAM are as follows:

1. Due to the significant difference in resistance between the different phases of GST, the PCRAM cell can be used to store the multi-bit information [47].

2. PCRAM is a scalable technology because as the feature density increases, it needs less current for the operations [48].

3. The SET and RESET latency of the PCRAM is larger than the STT-RAM and DRAM [20].

4. The endurance of PCRAM is bound to the limited number of writes. The current write endurance value varies in the range of 10^4 writes to 10^9 writes [49].

The commercial industries focus on PCRAM as a replacement of flash memory technology or to be used as the main memory. Different types of PCRAM chips at different technology nodes are manufactured and fabricated. For example, at 90nm node, Samsung electronics built the 512 Mb PCRAM chip with 266 Mb/s bandwidth [50]. Later, Samsung fabricated the 8Gb PCRAM chip at 20 nm technology with 40 Mb/s program bandwidth [51].

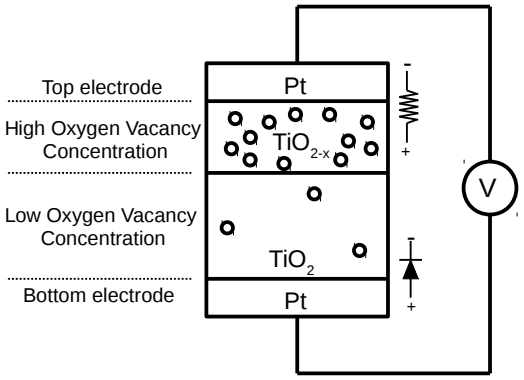


FIGURE 2.6: Representational view of ReRAM cell

2.1.2.3 Resistive Random Access Memory (ReRAM)

ReRAM [52] is based on the memristor technology where the resistance change depends upon the polarity, magnitude, and the duration of the applied voltage. Figure 2.6 shows the representational view of the ReRAM cell. The memristor-based ReRAM cell consists of two platinum electrodes with titanium dioxide (TiO_2) metal/oxide interference switches having different oxygen vacancy concentrations. Generally, the metal/oxide interference shows rectifying behavior with low doping and ohmic behavior with high doping [53]. In particular, the lower switch of perfect titanium dioxide (TiO_2) is electrically insulating and the upper switch, which has high having oxygen vacancy concentration (TiO_{2-x}) is conductive. The read and write operation of the ReRAM cell are described below:

Read Operation: To access the state of ReRAM cell, a small voltage is applied across the bit lines. This effect causes the current to be generated that can be sensed to detect the particular state of the cell.

Write Operation: To change the state of the cell, a large voltage is applied across the bit lines. To change the state of the cell to OFF, a negative bias voltage is used which increases the thickness of TiO_2, which in turn generates the insulating and high resistance ion path. The opposite case is seen in case of positive bias voltage for the ON state.

The other important properties of ReRAM cell are as follows:

1. The ReRAM memory technology is less mature than the PCRAM and STT-RAM memory technologies [54].

2. In the terms of scalability, ReRAM is more efficient compared to STT-RAM and PCRAM. The cell size of 10nm has been achieved and in future, the cell density of 4-5nm is predicted [55, 56].

3. The write endurance of ReRAM is limited to 10^5 to 10^{11} writes which is lesser than the STT-RAM and some of the prototypes of PCRAM [56, 20].

The ReRAM technology is still not as mature as other emerging NVMs and is currently under research. Recently, only Fujitsu and Panasonic are jointly working on the second generation ReRAM device [57].

2.2 Challenges to Employ Emerging NVMs in the Caches

The previous section reported the design concepts, essential operations and features of conventional charge-based memory and emerging NVMs. Table 2.1 shows the comparative analysis between the current memory hierarchy technologies: SRAM and DRAM and the emerging NVM technologies: STT-RAM, PCRAM, and ReRAM [37, 41, 20, 47, 55]. To draw the comparison, the following characteristics are used:

1. **Cell Size:** The cell size of the memory cell, measured in terms of feature size (F^2).

2. **Non-Volatile:** This characters is used to explain whether the memory technology is non-volatile or not.

3. **Endurance:** It is the total number of write operations that the memory cell can entertain before it eventually wears out.

4. **Read Latency:** It is the time consumed to perform a read operation in the memory cell.

Features	SRAM	DRAM	MRAM (STT-RAM)	PRAM	ReRAM
Cell Size	>100F^2	6-8F^2	37F^2	8-16F^2	>5F^2
Non-Volatility	NO	NO	YES	YES	YES
Endurance	$>10^{15}$	$>10^{15}$	10^{15}	$>10^8$	$>10^5$
Read Latency	<10 ns	10-60 ns	<10 ns	48 ns	<10 ns
Write Latency	<10 ns	10-60 ns	12.5 ns	40-150 ns	~10 ns
Dynamic Energy	Low	Medium	Low for Read High for Write	Medium for Read Very high for Write	Low for read high for write
Static Energy	High	Medium	Low	Low	Low
Maturity	Product	Product	Advance Development	Advance Development	Early Development
Retention	As long as voltage applied	<<second	>10 yr	>10 yr	>10 yr
Multi bit	1	1	2	>2	>2

TABLE 2.1: Comparative analysis of different memory technologies

5. **Write Latency:** It is the time consumed to perform a write operation in the memory cell.

6. **Dynamic Energy:** It is the energy spent during the read/write operation.

7. **Static Energy:** It is the energy consumed during the idleness of memory devices. This energy also includes the energy spent to retain the data correctly.

8. **Maturity:** This characteristic demonstrates that the current memory technology is in the early stage or later stage of development or it is commercially available in the market.

9. **Retention:** It is the ability of a memory cell that how long it can hold the data correctly without performing any refresh operation.

10. **Multi-bit:** It is the ability of a memory cell to keep the multi-bit information.

As it can be easily seen from the comparative analysis that the NVM technologies consume considerable write energy and latency, and suffer from weak write endurance. Next we will illustrate the challenges of NVM technologies when employed in the cache.

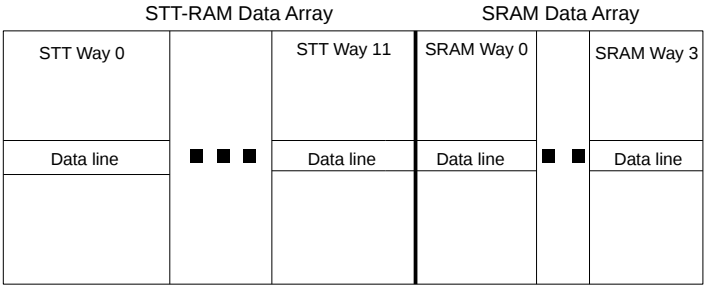


FIGURE 2.7: Hybrid cache bank organization

2.2.1 Challenges related to Write Operation

To counter the costly write operations, researchers make use of the concept of Hybrid Cache Architecture (HCA) [27, 28] which is described below:

Hybrid Cache Architecture (HCA): In HCA, the cache architects use the best characteristics of each memory technology. In particular, in cache bank, with a large number of NVM ways, a small number of SRAM ways are incorporated. Figure 2.7 shows the organization of a hybrid cache bank. The use of the SRAM ways in the HCA is to entertain most of the write operations in a cache bank; thereby, it saves the NVM region of the bank for writes. Block placement in such an architecture is the key challenging issue, as placing the appropriate block in the proper regions lead not only to the lesser write energy consumption but also improves the performance that is degraded due to costly write operations. Hence, in this context, different kind of block placement approaches have been proposed that take into account migrations and prediction of the block, reconfiguration, and partitioning of the cache. The next section illustrates all such policies. Apart from the block placement policy, the other challenge that debases the employability of HCA is the increase in the miss rate. This increase in miss rate in the HCA is mainly due to two reasons: (1) Irregular sized partitions of the cache. (2) Placement of a larger number of write-intensive blocks in the limited sized SRAM region. Table 2.2 presents the percentage increase in miss rate by one of the existing hybrid cache architecture: RWHCA [27] against the baseline pure NVM cache (which uses LRU as a replacement policy). The conclusion that can be drawn from the table is that due to larger miss rate in the smaller sized caches, the performance gain is not as much as expected with the HCA.

Work-load	PARSEC v2.1						SPEC CPU 2006			
Cache Config.	Cann	Ded	Fluid	Freq	Stream	X264	Mix1	Mix2	Mix3	Mix4
1MB	5.46%	15.3%	8.65%	18.3%	3%	6.8%	8.74%	7.82%	5.30%	12.30%
2MB	3.17%	11.7%	4.77%	16.25%	2.42%	5.36%	7.80%	6.23%	4.51%	10.32%
4MB	2.32%	10.8%	4.66%	15.7%	2.3%	4.46%	7.37%	5.40%	3.90%	8.51%
8MB	2.69%	10.3%	4.55%	14%	2.04%	3.27%	6.35%	4.01%	3.13%	5.60%

TABLE 2.2: Percentage increase in miss rate by RWHCA against baseline STT-RAM

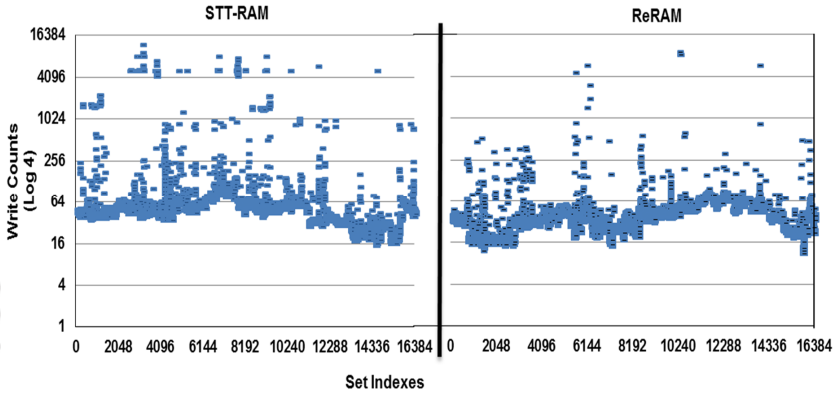


FIGURE 2.8: Write counts across the cache set for the baseline STT/ReRAM caches

2.2.2 Challenges related to Weak Write Endurance

The write endurance of the cache is defined as the total number of write operations that a memory cell can entertain before it breaks down. In the CMPs, different levels of the cache incur different numbers of writes. For instance, the upper-level cache experiences larger number of writes as compared to the last level caches (LLC). With a lesser amount of writes in the LLC, there is a good possibility that the writes are distributed unevenly inside the cache, which in turn generates write variation. In cache architectures, the researchers classify write variation into two categories [29]:

1. **Inter-Set Write Variation:** The inter-set write variation is caused due to the uneven write distribution across the cache-sets. In particular, some of the cache sets inside the bank incurs a large number of writes as compared to other sets. The figure 2.8 shows the presence of inter-set write variation in a cache bank. As can be seen from the figure, there is a non-uniform write

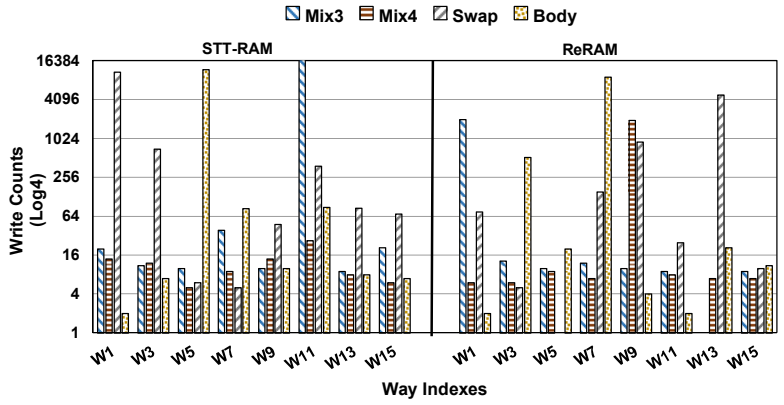


FIGURE 2.9: Write counts inside the cache set for the different workloads in the baseline STT/ReRAM caches

distribution across the cache sets. Such uneven distribution of writes results into wear out of heavily written sets faster than the lightly or moderately written ones.

2. **Intra-Set Write Variation:** The intra-set write variation is the variation due to distinct write counts of the blocks inside the set. Here, some of the blocks inside a set entertain larger number of writes as compared to other blocks in the set. Figure 2.9 reports the existence of intra-set write variation inside the cache sets. As can be seen from the figure, there is a variable write count among the different ways of cache sets. Such non-uniform write distribution leads to the early breakage of heavily written blocks as compared to lightly or moderately written blocks.

The two write variations: Inter and Intra-Set, mentioned above, are measured with the help of coefficients. Equations 2.1 and 2.2 present these coefficients: (i) $InterV$: measures the average coefficient of variation across the cache sets (ii) $IntraV$: measures the average coefficient of variation inside a cache set [29].

$$InterV = \frac{1}{Write_{avg}} \sqrt{\frac{\sum_{k=1}^{S} \left(\sum_{l=1}^{A} \frac{W_{k,l}}{A} - Write_{avg} \right)^2}{N-1}} \tag{2.1}$$

$$IntraV = \frac{1}{S.Write_{avg}} \sum_{k=1}^{S} \sqrt{\frac{\sum_{l=1}^{A} \left(W_{k,l} - \sum_{m=1}^{A} \frac{W_{k,m}}{A} \right)^2}{A-1}} \tag{2.2}$$

In these equations, A implies cache associativity, S represents the number of cache sets, $W_{k,l}$ is the write count in the cache set k and way l and $Write_{avg}$ is the average number of write counts in a cache bank.

Along with the weak write endurance, in the actual execution environment, the lifetime of the NVM LLC is further affected by these two write variations as mentioned above. The lifetime of the cache is defined as follows:

Lifetime: The lifetime of the caches can be defined either as raw lifetime or error tolerant lifetime [29]. The raw lifetime is determined by the first failure of the cache line. Whereas, the error tolerant lifetime is measured with the raw lifetime and the error recovery methods.

In this dissertation, we have used raw lifetime which is the basis of an error tolerant lifetime.

With respect to write variations and write count, the raw lifetime of caches can be determined by either of the following two methods:

1. With respect to write counts, the lifetime is the inverse of the maximum write counts on the block of the cache [31].

$$LI = \frac{1}{\forall_{k=1}^{S} \forall_{l=1}^{A} \max(W_{k,l})} \tag{2.3}$$

2. Concerning the write variation, the lifetime is calculated by considering the three important factors: (i) The coefficient of Intra-set write variation, $IntraV$ (ii) The coefficient of Inter-set write variation, $InterV$ (iii) Average

	PARSEC v2.1					SPEC CPU 2006			
Workload	**Body**	**Cann**	**Dedup**	**Swap**	**X264**	**Mix1**	**Mix2**	**Mix3**	**Mix4**
STT-RAM									
Ideal Lifetime	41.8K	3.8K	4.8K	14.3K	8.7K	3.21K	7.44K	14.9K	25.5K
IntraV	328.7%	57.4%	136.4%	361.4%	246.3%	21.6%	95.4%	223.6%	191.9%
InterV	450.7%	32.9%	66.4%	137.4%	325.3%	15.7%	213.9%	152.2%	96.5%
Baseline Lifetime	38	19.2	30.9	41.3	8.65	47.3	6.33	25.8	45.7
ReRAM									
Ideal Lifetime	2.12K	99.5	795.4	6.37K	462.2	185.4	175.2	565.5	666.2
IntraV	191.4%	49.37%	75.56%	396.6%	97.47%	11.25%	26.4%	80.6%	40%
InterV	332.8%	20.4%	161.9%	961.6%	50.9%	17.57%	17.1%	34.54%	17.45%
Baseline Lifetime	0.61	0.51	0.53	1.17	1	1.16	2.22	3.03	3.02

TABLE 2.3: Lifetime (in years) comparison analysis for the different workloads in the ideal STT-RAM/ReRAM and the actual STT-RAM/ReRAM based caches

write count in a cache bank [29].

$$LI = \frac{Write_{avg\_base} * (1 + InterV_{base} + IntraV_{base})}{Write_{avg\_pt} * (1 + InterV_{pt} + IntraV_{pt})} - 1 \tag{2.4}$$

In the above equation, the $Write_{avg}$ is the average number of write in a cache bank. Whereas, the *base* and *pt* used in the subscript with each term represent the metric value for the baseline and the proposed technique.

Table 2.3 reports the effect of the write variations on the lifetime of different levels of non-volatile LLC. The conclusion that can be drawn from the table is that compared to ideal caches (where the writes are uniformly distributed), the effect of the write variations on the lifetime (ideal lifetime for the ideal caches, baseline lifetime for the actual cache) of actual NVM cache is significant. The values reported in the table is computed by considering the write endurance value of STT to 4 x 10^{12} writes (considerably large) and the write endurance value of ReRAM to 10^8 writes. However, the recent chip fabricated from Samsung and Intel report the write endurance value of STT to 10^6 write cycles [44, 45] that can affect the lifetime further.

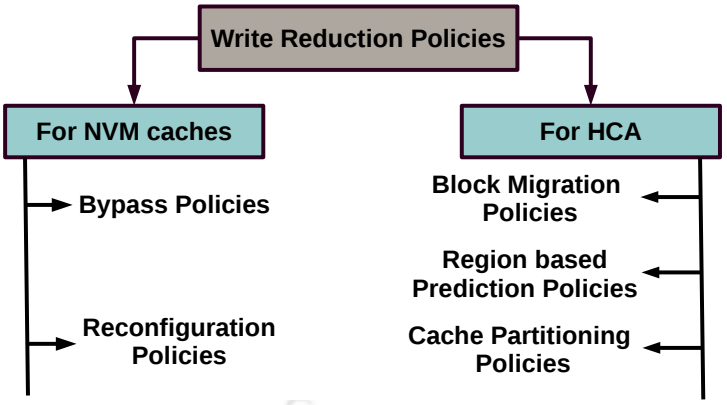


FIGURE 2.10: Classification of write reduction techniques based on the type of caches

2.3 Reducing the Costly Write Operations

One of the major hurdles with the employment of the NVM cache is the costly write operations. To mitigate the expensive write operations with the pure NVM caches, the researchers proposed the different bypass and the reconfiguration strategies. Other than these strategies, another way that has been employed by the researchers is the use of Hybrid Cache Architectures. From the earlier discussion in the context of HCA, it can be stated that the block placement is the major challenging issue to reduce the impact of costly write operations. Over the previous years, many block placement techniques have been proposed that fall into different sub-categories like prediction, migration, cache partitioning, and reconfiguration. Figure 2.10 classifies all state of the art write reduction techniques in the pure NVM and the HCA cache into different sub-categories. Below we discuss them one by one.

2.3.1 Migration Policies

The key objective of migration based techniques is to migrate the block to the appropriate region of the hybrid cache based upon the run-time accesses from the upper-level cache. The first policy that comes into this category is **Read Write aware Hybrid Cache Architecture (RWHCA)** proposed by Wu et al. [27, 28] in 2009. In RWHCA, the hybrid cache is separated into two regions: Read region made up of STT-RAM and the write region made up of SRAM.

The initial placement of the block into these regions is based upon the type of access that causes a miss. If it is a read miss, the incoming block is placed in the (read) STT region. Otherwise, for a write miss, the incoming block is placed in the SRAM (write) region. Also, in this policy, with each cache block, a 2-bit saturating counter is added that counts the region-wise access. Any disproportion of read or write accesses in any of these region results in the migration of the block from one region to another. The policy shows the 55% power reduction and 5% IPC improvement over the baseline SRAM and STT caches, respectively.

Li et al. [58] proposed a micro-architectural mechanism for the different write patterns of hybrid LLC. In this approach, the bank of the Hybrid LLC is either made up of SRAM or of STT-RAM. The block placement here is as same as the RWHCA. All the store miss blocks are placed in the target SRAM bank near to the requesting core, and all the load miss blocks are placed in the private STT-RAM bank of a core. Here, the migration of the write-intensive line from STT-RAM to SRAM is initiated when two consecutive writes or there cumulative writes are entertained by the block. On the other hand, the migration of the block from SRAM to STT-RAM is performed by two ways: Active and Lazy, based upon the position of the migrated block. If the block is fetched from the lower level of memory, then the active migration is triggered only after a read hit to a block. Otherwise, if the block is swapped from the STT-RAM, then the lazy migration is triggered after two read hits to the block.

In 2012, Chen et al. [59] proposed a static and dynamic approach that is based on the compiler hints for the block placement in the different regions of the hybrid cache. Here, any misinterpretation and misprediction in the access leads to the migration of block from one part to another in the hybrid cache. Guo et al. [60] proposed a wear resistant hybrid cache block placement approach where initially the cache line is placed in the SRAM region. During the lifetime of cache block, based on the different cache accesses, the line is categorized into Dead on Load (DoL) and Write Intensive (WI) Line. When the line is evicted from the SRAM, all the lines other than the DoL and WI are migrated to the non-volatile region.

Later in 2014, an adaptive block placement and migration policy is proposed in [61]. In this work, they categorized the block based on type of writes: Prefetch Write, Demand Write, and Core Write. The placement of the block is based upon the access patterns and the type of the writes. Whereas, the migration decision is taken by using the predictor. In particular, a block fetched due to prefetch miss will be directly placed into the SRAM region. Upon eviction, the predictor is used to check whether the evicted block is dead or not. If not, the block is migrated to SRAM. In case of a core-write miss, the block is written directly to the main memory. On the other hand, in case of a hit in STT, the possibility of future write burst to the block is checked with the help of predictor, and the block is migrated accordingly to SRAM. In case of a demand miss, the predictor is accessed to check if the incoming block is dead or not. If it is, the bypass operation is performed. Otherwise, the block is placed in the STT.

Wang et al. [62] propose a dynamic cache reallocation strategy to the different partitions of L1 based hybrid cache. Here, the cache blocks are transferred between the two regions by using the two mechanisms: Immediate Transfer and the Delayed Transfer. In immediate transfer, for the remote read operation to SRAM block, the block is transferred to STT-RAM, and for the remote write operation to STT-RAM block, the block is transferred to SRAM. Whereas, in delayed transfer, until two reads to SRAM block or two writes to STT block are entertained, the block is not transferred.

The other recent and the notable works over the past three years in the context of migration are reported in [63, 64]. In [63], the replacement policy of the cache is modified and it partitions the replacement stack into two regions: Reserved and Victim. The decision to place and migrate the cache lines from the different part of hybrid main memory into these region of replacement stack is based upon the Average Memory Access Time. Whereas, in [64], a priority based data migration to reduce the migration jitter for the frequently accessed data is given for the 3D based hybrid cache. In this approach, the data block is migrated between and in the layers with the priority in the X direction followed by the Y and then the Z direction. The policy reduces power consumption by 34.5%.

All the above block placement techniques are performing the migration of the block from one region/layer to another. The movement of the blocks between the regions/layers consumes extra energy as well as impacts the performance of the cache. As the applications running on the CMPs have variable behavior that can also change the block access behavior in the cache. For such cases, the massive migrations will nullify the benefits obtained by the existing techniques.

2.3.2 Prediction Policies

The role of the prediction policies is to predict the write behavior of the cache line and accordingly place it into the different regions of the hybrid cache. The first policy discussed in this subject is proposed in 2012 by Quan et al. [65]. Here, they manage the hybrid cache using a prediction table. By using the prediction table, they categorized the line into frequently written, less written, and dead cache line. During the execution, they assumed that the writes in the two consecutive stays of the cache line are probably equal. In case of a cache miss, the write frequency of the line in the previous visit is checked, if the incoming line is frequently written, the line is placed into the SRAM by replacing the line with the less written line. The replaced line from the SRAM will be placed in the STT region of the hybrid cache if it is predicted to be not dead. Otherwise, if the incoming line is less written or a dead line, then it is placed in the STT region.

In 2013, Ahn et al. [30] proposed a **write intensity prediction** technique for the data block brought down by the load miss. In this work, the prediction decision is taken by establishing the relation between the write intensity of the block and the instruction that incurs the cache miss. In the block placement approach, all the blocks loaded due to store miss and all the instruction blocks brought down by the load miss will be placed in the SRAM and the STT-RAM region, respectively. On the other hand, all the other blocks will go through the write intensity prediction and get appropriately placed into the STT/SRAM region by using the static write intensity threshold. Later in 2016, they proposed the extension of this work, where the write intensity threshold is changed dynamically and is decided based

upon the characteristics of the application running on the system [66]. Here, the concept of set sampling is used to reduce the storage overhead for storing metadata information used for prediction. The proposed technique shows 31% savings in energy consumption over the existing work.

A reuse distance based prediction scheme is reported by Kim et al. [67] for exclusive caches. On the basis of reuse distance, the cache line is predicted to be near reuse or far reuse. Based on predicted reuse distance, the block evicted from the upper-level cache is placed into the different region of the hybrid cache. In particular, all the evicted blocks predicted to be near reuse are placed into the SRAM region. In case, if these blocks are not accessed in the SRAM region for longer period and get evicted, the line gets migrated to STT-RAM. On the other hand, those blocks predicted to be far reused are placed into the STT region only if space is available. Another reuse distance-based scheme for 3D based DRAM and STT-RAM hybrid cache architecture is presented in [68]. In their work, with the help of reuse distance, the write probability is calculated. The block with the highest likelihood for the writes in the STT region will get swapped with the block with the lowest probability of writes in the SRAM region.

A recent noteworthy work that integrates there memory technology (DRAM, volatile STT-RAM and non-volatile STT-RAM) in a cache bank is presented in [69]. In this tri-regional hybrid cache, the read request follows the accessing order from non-volatile STT to volatile STT and then to DRAM. On the other hand, the write request follows the accessing order from DRAM to volatile STT and then to non-volatile STT. In their work, to reduce the time taken for the tag search operation, a data prediction table is added to predict hit and miss. The proposed technique saves average static and dynamic energy by 36% and 16% respectively.

All of the techniques mentioned above will heavily depend upon the accuracy of the employed predictor. However, the result of the predictor will change based on the application dynamic characteristics. In the case of correct prediction, the block is placed in the correct region, and this leads to saving in writes as well as

energy. However, in the case of misprediction, the block gets placed in the wrong region, and this leads to larger write energy, as well as impact on the performance.

2.3.3 Bypass Policies

As the name suggests, these policies detour a certain amount of data blocks by not writing them in the pure non-volatile cache. The first policy discussed in this category is Obstruction Aware Policy (OAP) presented by Wang et al. [70]. The OAP identifies those processes that obstructed the other running processes in the multi-core system and bypass their data from the NVM based L3 cache upon a hit or miss except for read hit. OAP reduces energy consumption by 64%.

Ahn et al. [71] illustrated a dead write based bypassing scheme called DASCA. In this work, they classify the write blocks into three categories: dead on arrival fill, dead value fill and closing writes. DASCA identifies such blocks and bypasses them from LLC by employing two types of bypass techniques: Upward Bypass (from main memory to upper level cache) and Downward Bypass (from upper level cache to main memory). The bypasses due to dead on arrival fill and dead value fill will fall into the category of the upward bypass. On the other hand, the bypasses incurred due to closing writes will come into the type of the downward bypass.

In 2014, a statistical based cache bypassing technique for (NVM based) L2 cache from L3 cache is reported by Zhang et al. [72]. Here, the bypass decision is taken based on the Distance Reuse Count (DRC). If the DRC of the particular block is lesser than the threshold (that changes dynamically), then the block is bypassed from the L2. Further, in 2016, to strengthen their statistical bypass decision, they integrate the core and group-based techniques where each cache line has a different bypassing depth in a cache bank [73]. In addition to this, they extend the bypassing mechanism for writing back the cache line from L2 to main memory without inserting back to the L3 cache.

Other recent noteworthy works presented in this context of bypass over the last three years are presented in [74, 75]. In [74], the writes in the non-exclusive non-volatile LLC are reduced by writing only the sub-block from the upper-level cache, thereby bypassing the writing of the other sub-blocks. To facilitate this process, they maintain a pattern history table that counts the miss prediction. In another work, a cache bypass technique is reported to relive the write congestion for serving the other request by maintaining a balance between the costly write latency and the hit rate loss [75]. Here, in this policy, to make a bypass decision, three factors: liveliness, write fraction and request bandwidth are considered.

The bypass policies discussed here reduce a significant amount of writes in one particular level of cache made of NVM technology. However, these techniques lead to a large number of write accesses, miss counts, and energy consumption of the next level of the memory hierarchy. Also, the bypassing technique is only limited to non-inclusive and the exclusive caches. In particular, it does not apply to strictly inclusive caches. Thus, an appropriate way of using bypassing at a particular level of cache is required.

2.3.4 Partitioning Policies

The role of the partitioning techniques is to logically partition the cache and use a separate partition for the block placement. These policies are intended for Non-Volatile caches, hybrid caches, as well as for optimized NVM based hybrid main memories.

In 2014, Lee et al. [76] proposed partitioning policies for the hybrid cache. In their work, they place the block to the different region based on the decision of utility based cache partitioning. In particular, all the store misses are placed in the SRAM region, and all the load misses are placed in either SRAM or STT-RAM based upon the result of the partitioning scheme. This scheme aims to reduce the miss rate by periodically changing the partition size allocated to the core.

Meanwhile, in 2015, Lin et al. [77] presented partitioning and the access aware policies to balance out the deranged writes and the wear out of STT-RAM in CMPs. In this scheme, the partition algorithm monitors the write pressure of each partition and accordingly decides to allocate and deallocate the number of ways of SRAM or STT-RAM to a core.

Hybrid memory Aware Partition technique, HAP presented by Wei et al. [78] maintains the appropriate count of the blocks from each region of hybrid memory by partitioning the LLC. HAP also has taken into consideration the eviction cost of dirty NVM writes from the LLC to memory, by using the 2-Chance technique. In HAP, to maintain the partition size, two range of threshold T_{high} and T_{low} is kept. On a miss, if the NVM count in the destination set is larger than the T_{high} and the incoming line is NVM line then the NVM line is evicted. In the other case, if the NVM count in the set is lower than the T_{low} and the incoming block is DRAM block then the DRAM block is evicted from the cache. For the rest of the cases, the LRU line is removed from the cache set. HAP improves performance by 46.7% and reduces energy consumption by 21.9%. Another LLC partitioning technique reported by Bakhshalipour et al. [79] reduce the number of write-backs in the hybrid main memory by coalescing writes for the dirty lines by retaining them in the LLC for a longer period. In their work, they partition the LLC into the normal and dirty partition. Here, the normal partition of the cache behaves as the normal cache. Whereas the dirty partition holds the evicted block from the normal partition and uses cuckoo hashing [80] for the data placement; thereby, it saves the unnecessary write-backs to the hybrid main memory.

Recently, in 2018, K-part reported by El-Sayed et al. [81] groups the applications into clusters and shares the different sized cache partition between them. Here, the grouping of the applications into the cluster is decided based on the performance loss of application due to partitioning of the cache. A dynamic profiling based mechanism is employed to make such decisions.

The partitioning policies discussed above are used for the block placement in the hybrid cache, to balance the deranged writes in the NVM cache and to reduce

the writes in the hybrid main memory by partitioning the LLC. However, by partitioning the LLC, the miss rate is increased, and the performance is affected due to less residency and premature eviction of the blocks.

2.3.5 Reconfiguration Policies

Reconfiguration techniques dynamically change the cache characteristics by turning off the cache banks/ways, changing the capacity of the cache by bank concatenation and at the level of the block size. In addition, some of the cache reconfiguration techniques have also made changes in the retention time of the non-volatile cache to reduce the latency and energy consumed due to writes. In this context, many prior works [82, 83, 84, 85] have taken the advantage of relaxing the retention time to get substantial benefits for latency and energy. This subsection reports all such policies.

Wang et al. [86] reported a study for reconfiguring the NVM cache where differential write [87] and data inverting scheme [88] is integrated. In their work, they have also made the slowest portion of the cache drowsy. The drowsy part will wake up only when the data is provided in execute or swap operation.

Niknam et al. [89] presented a dynamic reconfigurable hybrid cache that turns on/off different memory technologies bank based upon the Average Memory Access Time and the network traffic. In their work, the application execution is divided into multiple equal-sized intervals, and in each interval, the AMAT is calculated to reconfigure the hybrid cache. In particular, if the calculated AMAT is lesser than AMATref, the SRAM bank which has low write access is reconfigured with the STT-RAM bank. On the other hand, if the calculated AMAT is more than the AMATref, the STT-RAM bank having little read accesses is reconfigured with the SRAM bank.

Chen et al. [90] report a dynamic counter based way on/off reconfiguration policy. The decision to turn on/off way is taken at the end of each reconfiguration interval. The ways are turned off only when the decay counter associated with each way

are saturated. To turn on the way, the tag array of the way are kept the power on, and it counts the hits to be expected when the ways are power on using the potential counter. When the value of the counter reaches a specific limit, the way is turned on.

Another cache reconfiguration technique proposed by Adegbija et al. [91] performed the cache tuning by shutting down the way/bank, concatenating the way and changing the size of the cache line. To tune the cache at different levels, the energy objective function is taken into consideration.

Kuan et al. [92] propose a dynamic run-time adaptable retention scheme for L1 cache. Here, four sets of STT-RAM are set up with different retention time. Based on the run-time behavior, the applications are mapped on one of the set by taken into consideration the EDP or miss rate at each interval. Another versatile run-time technique for the LLC proposed by Kuan et al. [93]. As same as the previous technique, the applications are mapped into one of the four clusters with different retention time. Here, in addition of this, the size of the cluster is dynamically changed at the level of bank and line size. In this proposal, for the cache tuning, the latency is used as an objective function, and for the retention time tuning, the EDP is used as an objective function.

The reconfiguration techniques discussed above save considerable amount of write energy and improve the system performance by applying the constraints in the cache configuration. However, dynamically changing the configuration and by turning on/off leads to extra energy consumption. This loss is mainly due to the discharging of accumulated charge in the stand-by mode.

2.4 Improving the Lifetime and the Endurance of Non-Volatile Cache

Another hurdle with the employment of non-volatile cache is the weaker write endurance. In real-time execution environment, this lower endurance affects the

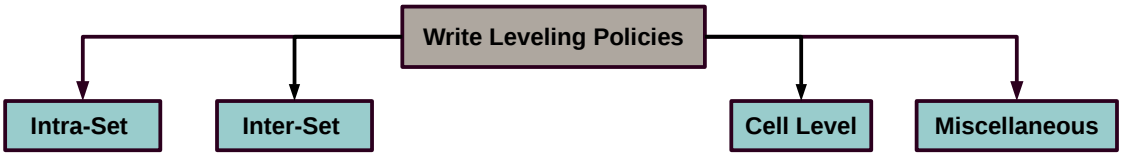


FIGURE 2.11: Classification on wear leveling techniques

lifetime of the non-volatile cache. Additionally, due to differing working set size and the run-time access pattern of the applications, the lifetime of non-volatile cache is affected by the write variations (categorized as an Inter and Intra-set write variation as reported in 2.2.2). To improve the lifetime and to mitigate the unwanted write variations, different kinds of wear leveling techniques have been proposed over the years. This section illustrates all such policies by categorizing them into many sub-levels. Figure 2.11 classifies these sub-levels in the wear leveling policies.

2.4.1 Intra-Set Wear Leveling Policies

The goal of intra-set wear leveling technique is to balance out the deranged writes inside the cache sets. Through this wear leveling, some of the cache blocks have been prevented to entertain more number of writes compared to another blocks inside the set.

In the year 2013, the first intra-set wear-leveling technique: **Probabilistic set line flush (Polf)** was proposed by Wang et al. [29] in i2wap. The policy invalidates a cache block after the fixed number of writes determined by the Flush Threshold (FT). For this, a counter is used which is incremented after each write to a cache bank. The selection of a block for invalidation is based on the probabilistic method rather than some deterministic ways. In their work, most of the time, the technique chooses the hottest data (write-intensive data) inside the cache set. Subsequently, the method flushes the hot data without changing its replacement information; thereby, the policy makes sure that the placement for the hot block on a subsequent miss will happen in another location in the cache set. To facilitate this process, the procedure uses two global counters and two registers.

Later, in 2014, **EqualChance** reported by Mittal et al. [31] added the counter: *numWrite* with each cache set. The *numWrite* is incremented with each write in the cache set. Once the *numWrite* counter reaches the threshold Υ, on next write access to a block, the transfer/swap operation with an invalid/clean cache line in the cache set takes place and the counter *numWrite* is reset. In case, if there is no clean/invalid cache line present in the cache set, the normal write operation is performed at the same location of the cache line. A technique LastingNVCache presented by Mittal et al. [94] associates the 4-bit write counter with each block in the cache. The counter is used to maintain the number of writes entertained by the block in a single generation. Once the counter reaches a specified limit, the write operation is skipped by invalidating the block without updating the replacement information. Another technique **WriteSmoothing** proposed by the Mittal et al. [33] partitions the cache into multiple modules of equal number of cache sets. Here, in this work, the write variation inside each module is reduced by turning off the hot sub-ways by transferring their data to cold sub-ways within the module.

Other most recent and noteworthy works in intra-set wear leveling are reported in [32, 95, 96, 97]. In [95], a technique called ENVLIVE is illustrated where the small storage called HotStore made of SRAM is added to store the write-intensive block of the cache. Only those blocks that incur the specific number of writes (determined by the λ) will be eligible for the placement in the HotStore. The technique named EqualWrite reported in [96] allows write redirection and swapping of the block based on the difference in the write counts of the cache line. The **Write-back Aware intra-set Displacement (WAD)** approach proposed by Jokar et al. [32] in Sequoia employs the counter with each cache set that increment on each write to a set. Upon a saturation of a counter, on a next write hit to a cache set, the policy displaces the block of the set to the victim cache line location by invalidating the victim line back to the main memory. A hybrid random replacement policy is presented in [97] that periodically switches the replacement policy between the traditional replacement policy and the random replacement policy for shuffling the actively written lines in a cache set.

The above discussed intra-set wear-leveling techniques reduce the write variation inside the cache set and improve the lifetime. However, the extra counters associated with these techniques will incur additional storage and area overheads to the system.

2.4.2 Inter-Set Wear Leveling Policies

The inter-set wear-leveling approaches aim to balance out the uneven write distribution across the cache sets. By this wear leveling, some of the cache sets will be prevented from getting worn out faster than the other cache sets inside the bank.

The first inter-set wear leveling technique: **Swap Shift** presented by Wang et al. [29] in i2wap changes the mapping of the cache sets after a fixed number of writes, determined by the Swap Threshold. In their work, the mapping of the set is adjusted by rotating the data inside the cache set. Chen et al. [98] presented an inter-set wear leveling technique that changes the mapping of the cache set at regular interval by performing an XOR operation between the content of the remap register and the set index of the block. Here, the content of the remap register is changed at the end of each interval of the application execution. A software controlled inter-set wear leveling approach that changes the cache color page mapping through write traffic in the cache is presented in [99, 100].

The other new works in the recent years in the inter-set wear leveling are reported in [32, 101]. In [32], a technique: Grouped Access Intra-Set Swapping is proposed in Sequoia that changes the cache set mapping between the heavily written and the lightly written set of the group with the help of the counters. In [101], Soltani et al. proposed an approach that partitions the cache into multiple clusters. During the execution, the clusters change their mapping to counter the inter-set write variation using the write intensity, mapping history, and the number of clean/invalid blocks.

The above-discussed approaches improve the lifetime and try to overcome the write variation across the cache set by changing the mappings of the sets. However,

the rearranging of the data according to the newly generated cache set mapping requires swaps or invalidation that consumes extra energy as well as impacts the performance.

2.4.3 Cell Level Wear Leveling

Apart from performing wear-leveling at the granularity of block, researchers have also tried to enhance wear-leveling at the memory cell level. This subsection discusses all such proposals.

Joo et al. [102] presented a technique to reduce the uneven write distribution inside the cache block by using the bit line shifter. The shifter is used to spread out the writes over the whole PCM cell for the cache. To aid this process, two registers: Shift Offset Register (SOR) and Shift Interval Counter (SIC) are used to record how many data bits are shifted in a cache block and the number of writes performed to cache block to update the SOR. A frequent data encoding scheme that largely reduces the number of redundant writes is proposed in [103]. The proposed technique is motivated by the fact that the hamming distance between the frequently generating codes is small. Thus, the data encoding scheme has the advantage to reduce the wear-out issue and improve the lifetime of the STT-based NVM cell. Another frequent pattern-based data encoding scheme to reduce the non-uniform write distribution is presented in [104]. In proposed work, the frequent data write pattern is categorized into two types: Deterministic and Non-deterministic. The frequent patterns are tracked by using dynamic profiling. During the application execution, these frequent data patterns get encoded, and their appropriate code bit is stored in the tag part of the STT-RAM cache. Recently, a word-level write variation reduction scheme that explores the narrow width data of the word to reduce the unnecessary writes in the STT-RAM cache is reported in [105].

2.4.4 Miscellaneous Wear Leveling

This subsection reports all other techniques that improve the write endurance of non-volatile memories either by using the hybrid cache or by exploiting the write reduction strategies.

A simple address space randomization technique that does wear leveling by the movement of line to its neighboring location is reported in [106]. To aid this process, two registers: Start and Gap and extra memory space Gapline are used. The Start register counts the number of times all the cache lines in the memory is relocated, and the Gap register counts the number of cache lines relocated in memory. A hybrid cache architecture based wear leveling technique: Ayush reported by Mittal et al. [107] migrates the write intensive data in the SRAM region of HCA. In this work, upon a write to the NVM region, the possibility of migration is checked by comparing the LRU age information of the NVM and the victim block in the SRAM. In case, if the SRAM contains an old data, the migration operation is performed. Sturkov et al. [108] presented an endure aware memory design that implements slower writes to reduce the stress on the NVM cell and to improve its lifetime. A ReRAM based NUCA architecture that does wear leveling in a performance conscious way by using the critical line predictor is reported by Kotra et al. [109]. Recently, an L1 cache based endurance aware data allocation strategy is proposed by Farbeh et al. [110]. The proposed work is motivated by the fact that the endurance of I cache is larger than the D cache in terms of number of writes. The strategy periodically makes alternate use of D cache as I cache and I cache as D cache.

2.5 Summary

With a large number of cores integrated on-chip, the basic building blocks of modern computing systems (CMPs) require multi-level on-chip caches. To reduce the off-chip memory access, large-sized on-chip LLCs are incorporated which occupy

a significant amount of on-chip area. Traditional caches made up of charge based memory technologies like SRAM/DRAM consume a lot of leakage power due to the continuous process scaling and fail to fulfill the application demands in terms of scalability. To mitigate this, computer architects have moved towards emerging NVMs and look them as alternate memory technologies in the cache hierarchy [111]. The gain obtained by using the NVMs is low leakage power consumption, high density, multi-bit storage capability and excellent scalability. However, by employing NVMs in the cache hierarchies it will suffer from costly write operations and weak write endurance; thereby it will impact the performance, energy consumption and the lifetime of the caches.

Over the previous years, many attempts have been made to counter the costly write operations of the NVM cache by using the reconfiguration policies and bypass techniques. Further, to reduce the expensive write operations in the NVMs, researchers use the best characteristics that each memory technology offers by the use of Hybrid Cache Architecture. In HCA, block placement is the most challenging issue so as to place the appropriate block in the proper region. In the context of HCA, different efforts have been made for the block placement using the block migration schemes, region-based prediction techniques, and by using the cache partitioning strategies.

To endure the cache from the write variations exhibited due to concurrent execution of the multiple applications, researchers proposed different strategies at different granularities of the cache. In particular, the write variation inside the cache set are mitigated by using the intra-set wear leveling techniques. On the other hand, the write variation across the cache sets is alleviated by inter-set wear leveling techniques. Also, instead of concentrating on wear-leveling at the block-level, the architects cope up the non-uniform write distribution inside the content of block by using cell level wear-leveling. Further, along with the wear leveling techniques, the endurance of non-volatile cache is enhanced by the write reduction schemes and by using the HCA.

Thus, the careful management of the writes in the NVM technologies can make them a suitable candidate in the cache hierarchy for an efficient hardware system.

भारतीय प्रौद्योगिकी संस्थान गुवाहाटी
Indian Institute of Technology Guwahati

Chapter 3

Reducing Write Cost by Dataless Entries and Prediction

In this chapter, we discuss the first contribution to the longevity enhancement of non-volatile cache. We proposed a data allocation policy that reduces the number of writes and energy consumption of the STT-RAM region in the Hybrid last level cache by considering the existence of private blocks. In addition to this, we employed a predictor that helps to redirect the write-backs from L1 to SRAM region of the hybrid cache; depending on the predicted reuse distance aware write intensity. The proposed work is evaluated with two different existing techniques in case of the dual and quad-core system.

3.1 Introduction

As discussed in Chapter 2, the block placement policy in the HCA is considered to be a challenging critical issue. Previously several strategies in HCA have been proposed, but as per our knowledge, none of the existing literature exploit the existence of private blocks. Private blocks are those blocks that are requested by a single core, and the cache controller serves the request with exclusive permission (i.e., for both read and write operation). Whereas, the block requested by multiple

cores is served with the shared or read-only permission to the requesting cores. In case of private blocks, for most of the time, the blocks in the L2 (or LLC) cache contain stale data [112, 113] (The conclusive evidence is given in section 3.2.1). For such kind of blocks, the actual worthy data is updated when the block is written back from the L1. In other words, data in the L2 cache is not required or needed until the L1 cache performs the write-back operation. When the private block is loaded into the L2 cache, the data part of the block is not stored in the data array of the cache. To maintain such dataless entries in the L2 cache, we make some changes in the conventional MESI protocol by adding some new states and their associated transitions. Also, our policy uses a predictor to decide on whether or not the first write back from the owner L1 cache to the dataless entries in the non-volatile region should be redirected to the SRAM region. The decision of our predictor is based on the reuse distance aware write intensity of the block. Further, in this chapter, we also present replacement policies for the different regions of the hybrid cache.

In this work, our proposed policy makes use of private blocks and avoids storing its data part in the non-volatile region of the LLC. We use STT-RAM as a non-volatile region of hybrid cache, although the policy can be easily extended to use PCRAM or ReRAM based hybrid caches.

The main contributions of this work are as follows:

- We consider the existence of private blocks in the HCA and avoid storing its data part in the non-volatile region.
- To maintain dataless entries in the STT region of the hybrid cache, we make changes in the conventional MESI protocol. These changes include the addition of new states and their associated transitions to the existing protocol.
- We propose a predictor to decide whether or not the first write-back from the owner L1 cache to the dataless entries in the non-volatile region should be redirected to the SRAM region. The decision of our predictor is influenced by the reuse distance aware write intensity of the block.

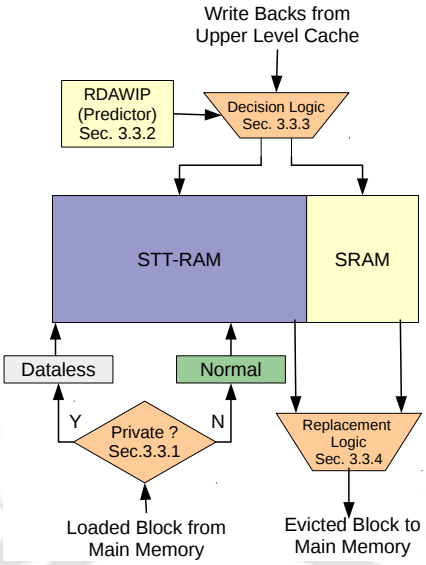


FIGURE 3.1: General overview of contribution in chapter 3

- We also propose a replacement policy for the different regions of the hybrid cache.
- The proposed techniques are evaluated against two existing techniques: Read Write aware Hybrid Cache Architecture [27, 28] and Write Intensity prediction technique [30]. Experimental results show the reduction of overall writes along with savings in energy.

Figure 3.1 presents the general overview of the proposed contributions in this chapter.

The rest of the chapter is organized as follows: Motivation and Background are presented in Section 3.2. Section 3.3 illustrates the proposed hybrid cache architecture along with the concept of set sampling. Section 3.4 discussed the experimental methodology. Results and analysis are presented in Section 3.5. Finally, we summarize this chapter in the last section 3.6.

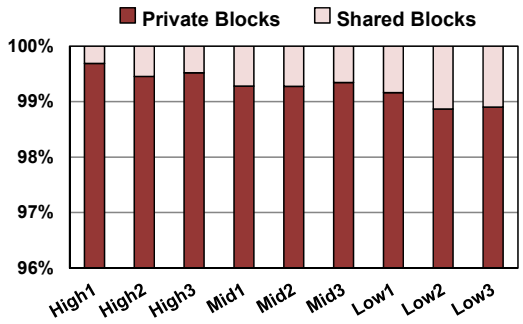


FIGURE 3.2: Percentage of private and shared blocks brought from the main memory on L2 miss.

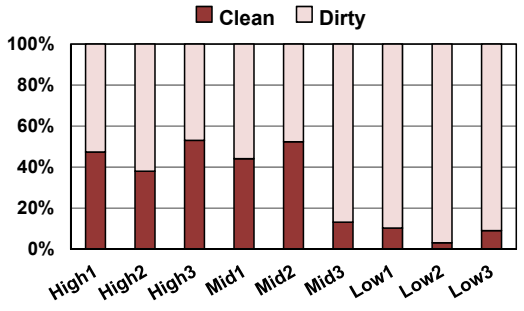


FIGURE 3.3: Percentage of blocks having exclusive permission which are clean or dirty at the time of replacement

3.2 Motivation and Background

3.2.1 Private Blocks

As stated earlier, we consider the existence of private blocks in this work. To measure the impact of private blocks on the cache, we conducted an experiment on 8MB L2 cache (Details about the experimental setup used in this motivation example are reported in Section 3.4). Fig. 3.2 shows the percentage of private and shared blocks loaded from the main memory on the LLC miss. From the figure, we can conclude that the single-core requests 98% of the blocks, i.e., these are private blocks. Whereas, the rest of the blocks are either instruction blocks or the blocks requested by multiple cores called shared blocks. Further, fig. 3.3 shows the percentage of the blocks having exclusive permission (i.e., private blocks) that contain dirty data at the time of replacement. As shown in the figure, on an average 50% of these blocks contain dirty data at the time of replacement. This shows that the blocks loaded with exclusive permission in the LLC contain worthless or stale data at some point in their lifetime. This inspires us to recognize the private blocks and avoid storing its data part in the non-volatile region of the LLC. In these dataless entries, the actual worthy data is present after the write-back operation from L1 cache to the L2 cache (i.e., the LLC).

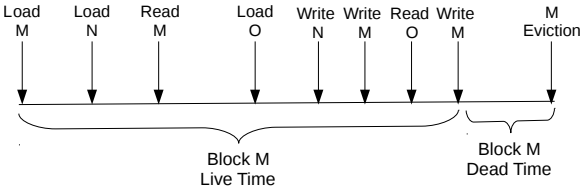


FIGURE 3.4: Reuse count example

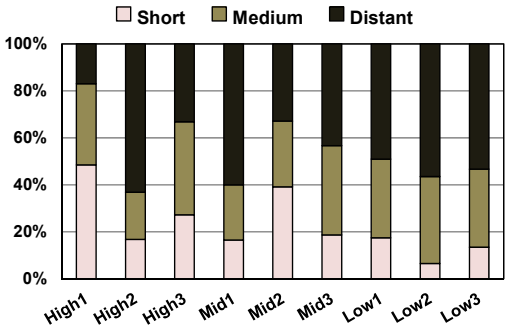


FIGURE 3.5: Collected reuse distance for different workloads

3.2.2 Reuse Distance

In this work, we design a predictor to decide whether the first write-back from the owner L1 cache should be redirected to the SRAM region of the hybrid LLC. The decision made by the predictor is based on the reuse distance and write intensity of the block being written back, by mapping the write-back block reference address as an index in the predictor table. The reuse distance of the block address is defined as the number of intervening accesses or the number of read or write operations on other blocks between the two consecutive accesses to this block during the live time of the block. As explained in Fig. 3.4, the reuse distance of the block M is 2. Based on the reuse distance of the block, we classified them into three categories: short, medium, and distant. Fig. 3.5 shows the percentage of the blocks falling into these three categories for different workloads.

3.2.3 MESI Protocol

To maintain dataless entries in the non-volatile region of hybrid cache, we make some changes in the conventional MESI protocol. These changes include the addition of some new states and their associated transitions. The MESI protocol contains four stable states: M (Modified), E (Exclusive), S (Shared), and I (Invalid) to maintain the cache coherence. The cache block is in state **M** when the block has exclusive permission (that means the block is readable and writable) and the core has modified it. For the block in state M, it is the only valid copy of

the block present on the chip. The block is in state **E** when it is not modified yet holds the exclusive permission. The state **S** of the block represents the block has read-only permission, and it is shared among multiple caches. The state **I** of the block represents the block which is not cached or not present. Furthermore, with each entry of L2 or LLC, the directory entry is associated. The directory entry maintains the following set of information:

1. **State field:** Different states of the L2 cache block is maintained within this field.
2. **Sharer list:** This field maintains the list of L1 caches which share the L2 cache block.
3. **Owner field:** Pointer to the owner L1 cache that holds the L2 cache block in exclusive mode.

3.3 Proposed Hybrid Cache Architecture

Our proposal is based on the modification of MESI protocol to include the existence of private blocks.

3.3.1 Basic Organization

Fig. 3.6 shows the organization of our proposed hybrid cache architecture. In our work, L2 cache (which is the LLC) is a hybrid cache that combines a large number of STT-RAM ways with a small number of SRAM ways [27, 28]. This asymmetry in the number of ways between two memories is to limit the static power dissipation by the SRAM and to redirect the limited number of write intensive blocks (limited number in the working set of the application) in the SRAM region (as already elaborated in section 2.2.1). Our data placement policy allocates dataless entries in STT region of L2 cache by sending the private blocks directly to the requesting L1 cache. Same as in a conventional SRAM cache, the tag array of the hybrid

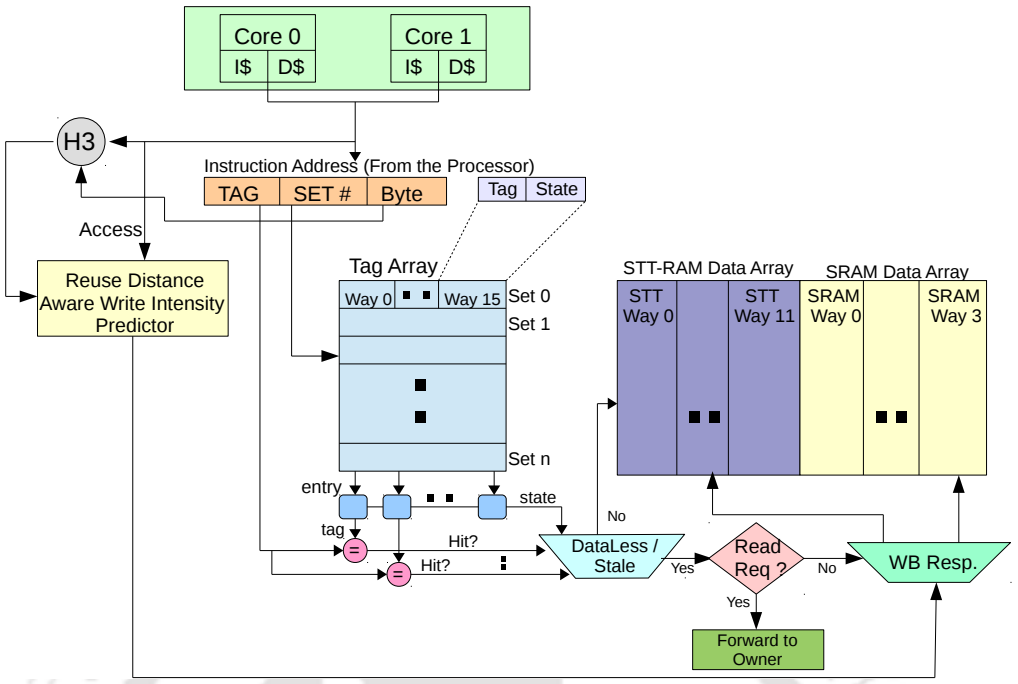


FIGURE 3.6: Overview of hybrid cache architecture along with the proposed block predictor

cache is made up of SRAM memory technology. In particular, if the X units of tags (made of SRAM) are required for the SRAM region, then 3X units of tags are needed for STT region and these tags are made up of SRAM. In the hybrid cache, the working of the lookup operation in the tag array is the same as in the traditional SRAM cache. In case of a miss from an L1 cache, the address of the instruction along with the block request is sent to the L2 cache. The tag lookup operation is performed on the L2. If the result of the lookup is a hit, the L2 cache controller checks whether there is data in the L2 cache. If data is present in the L2 cache either in SRAM or STT-RAM, the block request from the L1 cache will be directly served by the L2 cache. On the other hand, if data is not present in the L2 cache (i.e., dataless entry for a private block) or the data is stale, the request from the L1 cache will be forwarded to the owner L1 cache to fetch the block and then forwarded to the requester. In case, when the block is not found in the hybrid cache, the block request will be sent to the next level of the memory (i.e., main memory in our case). The Reuse Distance Aware Write Intensity Predictor shown in Fig. 3.6, and it is discussed in 3.3.2.

L1 Request or Response	Description
GETS	Request from L1 for Shared access
GETX	Request from L1 for Exclusive access
GET_INSTR	Instruction Request
PUTS	L1 replacing clean data
PUTX	L1 replacing Dirty data
WB_DATA	Data from L1
WB_CLEAN	Clean data from L1
DATA	Data from Main Memory
UPGRADE	Request from L1 for Exclusive from Shared access

TABLE 3.1: Events initiated by the local L1s

3.3.1.1 States added in the MESI Protocol

The main idea of the proposal is to identify the private blocks and store only the tag part when they are loaded from the main memory on an L2 cache miss. The actions needed to be taken for the loaded block according to the requests/response generated by different cores are explained in this subsection. Table 3.1 lists the request or response generated from the L1 cache along with its brief description. In this subsection, we discuss (1) The actions and states of the blocks when they are loaded from the main memory on an L2 cache miss. (2) The transitions of the blocks to different states, when they are replaced by the owner L1 cache or when there is a new request or response from other L1 caches. (3) Migration of blocks to SRAM, in case they become write-intensive. Note that here, the data is migrated to the SRAM on the second write-back operation to STT region; as we consider such blocks as write-intensive. All the cases are shown with the help of the MESI protocol state diagram. As we maintain the entries without data in the STT region of hybrid cache, we make some changes in the MESI protocol. The changes include the addition of some new states and their associated transitions. The details of all the cases mentioned above are described below:

Fig. 3.7(a) shows the actions and states of the block when they are loaded from the main memory. The given state diagram has two types of states: (1) Tag with Data state, and (2) Tag only state. A brief description of the states is given below:

- **I:** Invalid entry of L2 cache.

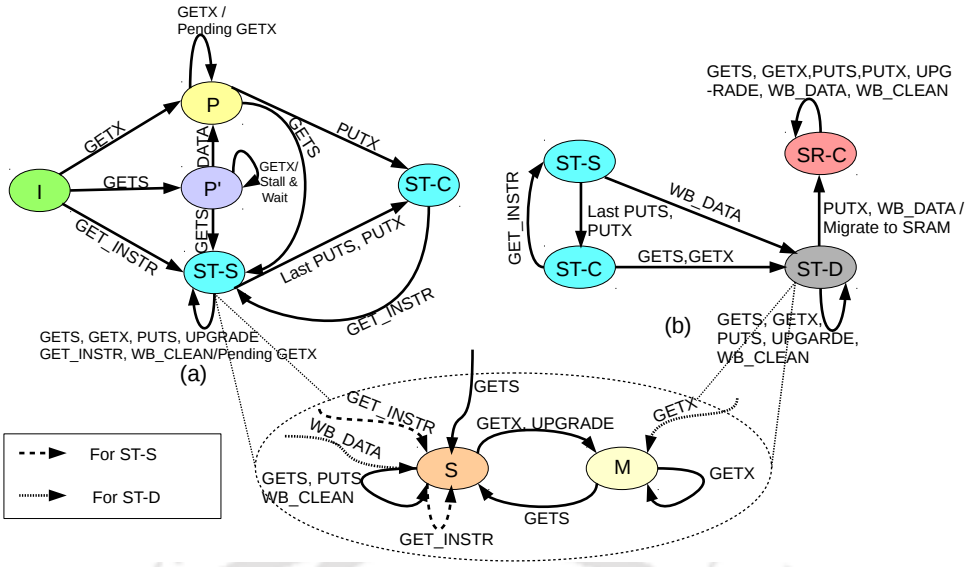


FIGURE 3.7: (a). State Diagram of STT region of the cache showing new states along with the associated transitions (b). State Diagram showing the migration process from STT-RAM to SRAM region.

• **ST-S:** An L2 cache entry that is shared between two or more cores or having read-only permission in the STT region of the cache. Note that it is a tag with data state.

• **P:** An L2 cache entry in the STT region that is held exclusively (both read/write permission) by L1 cache. Note that it is the tag only state.

• **ST-C:** An L2 cache entry in the STT region of the cache (without any owner or sharer(s)). Note that it is a tag with data state.

• **P':** The state acts as an intermediate state. The transition to this state happens when some core requests the block with read-only permission (GETS). If the current request is served without any intervention by another core or other L1 cache (Another read-only request from the other core), the L2 cache block is served to the requestor L1 cache along with the exclusive permission and the state of the block changed to P. Otherwise, if there is an intervening request (GETS) from another core while the block is being loaded from the main memory, the incoming block is served with the read-only permission to the requesting L1 caches. In this case, the state of the block is changed to state ST-S.

(a) ***Actions to be taken on L2 cache miss***

The description of transitions between the states according to the requests/responses generated by the L1 cache for the block that is **not present** in the L2 cache is described below:

- ***When the core or L1 request an instruction block (GET_INSTR):*** The request is sent to the next level of memory, and the incoming block is loaded in the STT region of the cache. In the directory entry, the state field is changed from I to ST-S. The requested block is sent to L1.
- ***When the block is requested exclusively (GETX) by L1 cache:*** The incoming block from the main memory will be allocated in the STT region of the cache. But in this case, the loaded block will not update the data array, and the array remains empty. The state of the block is changed from the state I to P. The requested block is sent to L1.
- ***When the block is requested with read-only permission (GETS) by L1 cache:*** The state of the block is first changed to intermediate P' state. Now, in this case, the final state of the block depends on whether there is an intervention from another L1 cache (Read Request from another L1 cache). If there is no intervention, the block is served exclusively to the requestor L1 by changing the state from state P' to P. Otherwise, a block is provided with the read-only permission to the requestor L1 caches. In this case, the state of the block is changed to state ST-S.

(b) ***Actions to be taken on a block replacement from L1 cache or on a request or response from the L1 cache to the block in the L2 cache***

When the block is placed into the L2 cache, there can be many changes in the state of the block. The changes are due to request or response generated from the L1 cache or when the block is replaced from the L1 cache. All these changes are described below:

- ***When the block is replaced from the L1 cache (Last PUTS or PUTX) and the state of the block in the L2 cache is ST-S:*** The write-back

operation is scheduled according to the dirty bit of the block. Once the write-back operation is performed, the state of the block is changed to state ST-C.

- ***When the block is evicted from the L1 cache (PUTX) and the state of the block in the L2 cache is P:*** The write-back operation is scheduled irrespective of the dirty bit of the block. In this case, the L2 cache entry is dataless, and so the data has to be written regardless of its dirty status. The state of the block is changed to state ST-C.
- ***When the L2 cache block is in state ST-S, and the sharers of the block are reduced to one:*** The state of the block remains ST-S. The block can either contain fresh or stale data, and it can handle both M and S state of MESI protocol. The way ST-S handles both M and S states is shown in the dotted ellipse in fig. 3.7.
- ***When the block is requested exclusively (GETX) by some other core and the state of the block in the L2 cache is P:*** In this case, the request from the other core will be forwarded to the owner of the block. Then, the owner L1 will send the data to the requestor L1 and invalidate its own copy of data. In this situation, the requestor will become the new owner of the block, and the owner field in the directory is updated accordingly. The state of the block remains P.
- ***When the block is requested with the read-only permission (GETS) by some other core and the state of the block in the L2 cache is P:*** The request from the other core will be forwarded to the owner of the block which in turn sends the data to the requestor. But in this case, instead of invalidating its own data, the owner L1 will keep the copy of data with shared permission. As the block is shared by more than one core, the write operation is performed on L2 cache as the cache entry did not contain any data previously. The state of the block is changed to state ST-S.
- ***When the block is requested exclusively (GETX) by some core and the state of the block is P' :*** The write request will be stalled and will be served later. The stalled request will continue to be stalled until the state of the block is changed to either ST-S or P.

- ***When the core or L1 request the instruction block which is residing in L2 cache with state ST-C:*** The instruction request will be served by providing the block to the requestor L1 along with the shared permission. The state of the block is changed to state ST-S.

(c) ***Migration of Block to SRAM***

Our policy migrates the block to SRAM region when the second write-back operation is performed on that block in the L2 cache. Fig. 3.7(b) presents the migration process from STT to SRAM. The description of the states used in the state diagram are presented below:

- **ST-D:** An L2 cache entry that resided in the STT region and is the potential candidate for migration. Note that it is the tag with data state.
- **SR-C:** An L2 cache entry residing in SRAM region with or without owner/sharer(s). The state SR-C is used here as an abstraction as it follows the normal MESI protocol for SRAM region. Note that the state is a tag with data state.

The actions to be taken while migrating the block are described below:

- ***When an L2 cache block having no owner/sharer(s) (state ST-C) gets a read or write request (GETS or GETX):*** In this case, the block becomes a potential candidate for migration due to prospective multiple write requests. The state of the block is changed to ST-D, and the data is sent to the requestor L1.
- ***When an L2 cache block receives the write-back response and the state of the block is ST-S:*** In this case, the write-back block becomes a potential candidate for the migration, the state of the block is changed to ST-D.
- ***When a block in the L2 cache gets a read or write request and the state of the block in the L2 cache is ST-D:*** The request will be treated similarly to state ST-S. In this case, the state of the block remains in ST-D, and it can handle both M and S state of MESI protocol, as shown in the dotted ellipse in fig. 3.7.

- ***When a block receives the write-back response (PUTX, WB_DATA) and the state of the block in the L2 cache is ST-D:*** In this case, the migration process of the block from STT to SRAM is executed in two steps. First, the lookup operation for the invalid entry is performed in the SRAM region. If there is no invalid entry, the LRU victim is selected from the same cache set in the SRAM region and the write-back operation is scheduled to the next level of memory for the victim. Second, the write-back response will be redirected to the available entry in the SRAM region by transferring the tag from STT-RAM to SRAM region. Afterward, the block will be invalidated from STT-RAM. Once the write-back operation is performed, the state of the block in the SRAM region is changed to state SR-C. The state SR-C can either act as an S or I (in case of WB_DATA) or M (in case of PUTX) state of MESI protocol. For simplicity purpose, we denoted this by state SR-C in the state diagram.

The benefits of these additional states (shown in fig. 3.7) is to represent the dataless entries (P) in the STT region and to show the states of MESI protocol in an abstract manner (I, ST-S, ST-C, ST-D, and SR-C) according to different regions of HCA. The overheads with the state diagram is to incorporate the dataless and intermediate state in the directory structure (that will add few bits and 5 cycles to search in the directory and the extra changes required to maintain the dataless entries) of the protocol. We have considered all these overheads in the simulations.

3.3.2 Design and Operation of the Reuse Distance Aware Write Intensity Predictor (RDAWIP)

The protocol elaborated in the previous section discussed the existence of dataless entries in the STT region of the L2 cache. During the second write operation to such entries, the block was declared to be heavily written block and migrated to the SRAM region. Several such blocks become candidates for migration and eventually move to the SRAM region. However, before relocation, these blocks

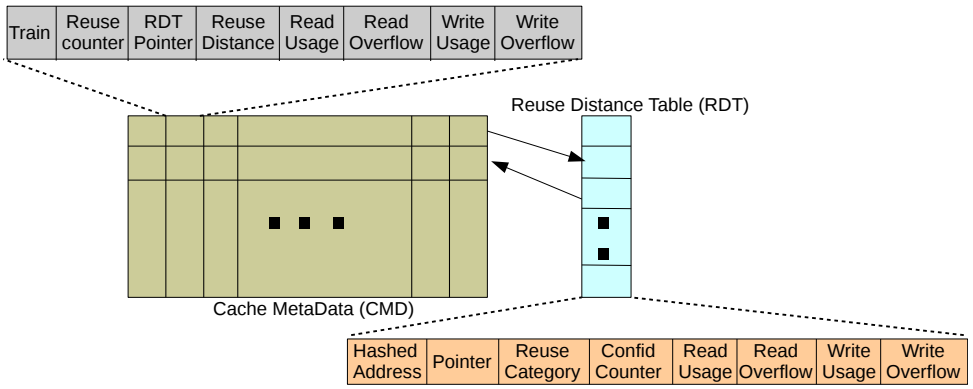


FIGURE 3.8: Organization of Reuse Distance Aware Write Intensity Predictor

incur one write to the STT region. One can avoid these writes if we can predict those blocks which eventually migrate to SRAM. To incorporate this mechanism, we propose a predictor based upon reuse distance and write intensity of blocks. Depending on the reuse category: viz. *short*, *medium* or *large* the decision to redirect the write to SRAM region is taken during the first write-back for such blocks. In other words, when a dataless block receives the first write-back from L1 cache, the predictor directs to redirect it to SRAM if it is predicted to be heavily written block. Otherwise, the block is written to STT region and later migrates to SRAM if it incurs more writes. The performance of this policy depends on the prediction accuracy, which is analyzed in the experimental section 3.5.5.

The Reuse Distance Aware Write Intensity Predictor (RDAWIP) mechanism uses two additional data structures: the Reuse Distance Table (RDT) and the Cache MetaData (CMD). The composition of these two data structures is shown in Fig. 3.8. Note that the two structures used in the predictor are made up of SRAM. The RDT is a table with a limited number of entries indexed using H3 hash function [114] on a subset of bits from the program counter (PC) and the byte offset from the memory reference address. The reason behind using the PC-offset combination is their high coverage and accuracy for different workloads [115, 116, 117]. Each entry in the RDT stores the reuse category and confidence counter apart from read/write counters like the Read Usage (RU), Read Overflow (RO) and the Write Usage (WU) and Write Overflow (WO). In particular, each entry of RDT

stores the read and write behavior of the hashed address according to the reuse distance.

The other data structure to be used in our technique is Cache MetaData (CMD). The CMD is a table that stores the metadata information of each block in the cache. The entry of CMD contains many fields such as: read/write counters (Read Usage, Write Usage, Read Overflow and Write Overflow), train bit, RDT pointer and Reuse information (Reuse counter and Reuse Distance). The description and the use of these fields are mentioned below:

- The use of read/write counter is to capture the read/write behavior of the associated cache block during the execution.
- The reuse information fields are used to store the reuse distance of the block.
- The RDT pointer in the CMD is used to link the RDT entry with the associated cache block.
- The train field is used to identify the cache entry involved in initialization/update the corresponding RDT entry.

The use of CMD entry is to populate, update, and verify the data stored in the RDT as per the access during the live time of the block. When the block is evicted, the entries in the CMD are initialized to zero or reset.

3.3.2.1 Initialization Phase

The initialization phase of the RDT is started when the entry mapped by H3 hash function with PC-offset combination is not found in RDT. In this case, a new entry is created in the RDT, and the CMD of the block is mapped to this newly created entry. The address whose PC maps to an entry in the RDT is used to update the values of the counters. In other words, that address/block is responsible for initializing the RDT for prediction.

Whenever the block that is linked to the RDT entry incurs a read/write request the corresponding counters in the RDT entry are updated. In case the count reaches saturation, its corresponding overflow bit is set. Note that the reuse information of the block is maintained in its CMD. The initialization phase of the RDT stops when the block linked to an entry in the RDT is evicted from the cache. At this time, the reuse information of the CMD is used to fill the reuse category in the RDT.

3.3.2.2 Usage or Update phase

During execution, several hashed-PC addresses will map to a given RDT entry. However, only one of them will be used to update the RDT. Note that if RDT block is under initialization phase, then other blocks that are mapped to the same entry will not be involved in the initialization phase. Once the initialization phase is over, the new block(s) mapping to the corresponding RDT entry will copy the data from RDT in the CMD. With each read and write request to the mapped block, the respective counters in the CMD are decremented accordingly. In the case, when the counter of the CMD reaches the saturation, and their associated read/write overflow bit is set then this block is assumed to be read/write-intensive and hence no more changes are required in RDT.

However, there is a possibility of updating the RDT when the read/write overflow is not set. In this case, when the count of read/write usage counter of the CMD exceeds the predicted value (becomes zero), we start the update phase. For this, we set the train bit and start updating the read/write counts (incremented or set the read/write overflow in case the count reaches saturation) of the RDT. When the corresponding block is evicted, the RDT has the new information of the read/write count as per the behavior from the respective evicted entry. The removed CMD entry also maintains the reuse distance information that helps to verify the reuse category of the RDT. If the reuse category matches with the reuse information, the confidence counter is incremented. Otherwise, decremented. In

case the confidence counter becomes zero, we again categorize the RDT reuse category as per the reuse information stored in the CMD.

3.3.3 Prediction of the First Write back from L1 to L2

Our policy predicts the region of the hybrid cache to which the first write-back (from the owner L1 cache) is redirected. The redirection scheme is governed by the following cases:

1. Prediction is not used
   - For blocks having a tag with data entries in STT region of the cache. In this case, the write-back operation is performed only in the region in which the data resides.
   - For the block involved in initialization of the RDT (i.e., train flag =1 in CMD).
   - For the block mapping to an RDT entry with confidence counter value zero.
2. Prediction is used
   - For blocks with the dataless entry in STT.
   - For blocks having RDT entry with train flag = 0.
   - For blocks having RDT entry with the value of confidence counter greater than zero.
3. If the prediction is used for the block, the decision is taken with the help of write usage counter, write overflow bit and the reuse distance of the block in the RDT entry. For short and medium reuse distance, the block is redirected to SRAM only when the value of write usage counter is 2 or more. While in the case of long reuse distance, the block is redirected when the write overflow bit is set [1].

[1]We conducted extensive profiling for the different sets of values of write usage counter and with different reuse distance. And, accordingly, we selected the most stable values.

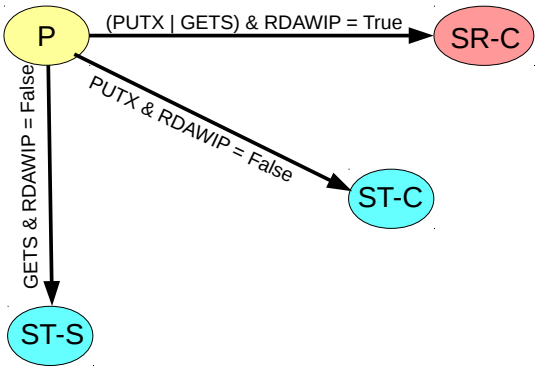


FIGURE 3.9: State Diagram showing the modified transactions with respect to results of the predictor

In case, due to inaccuracy by the RDAWIP, if the block is not redirected to SRAM on the first write-back, it will be migrated by the protocol (during its second write back), by considering it as write-intensive (due to prospective multiple write requests). Note that the only overhead in this is the first write back incurs at the STT region of Hybrid Cache. However, there are very few times that the predictor makes an incorrect predictions (including overestimation and underestimation, as it is evident from the sections 3.5.5 and 3.5.8.5).

Figure 3.9 shows the modified transactions in response to the result of the RDAWIP for the first write-back for the dataless entries having the state P.

3.3.4 Augmenting the Replacement Policy

Our proposed prediction method can also be used to augment the replacement policy to improve the decision of conventional LRU. The new replacement policy makes use of the predictor field to prioritize the cache line for early eviction. The usage counter and reuse distance stored in the cache metadata and RDT decides the prioritization of the line. The blocks which have RDT entries already trained (i.e., train bit in CMD is not set) are used for predicting the early replacement. If RDT anticipates that the block has completed its read and write actions, then it can be considered for victim selection. In other words, blocks having read and write overflow not set and whose usage counters (read/write) zero are considered to

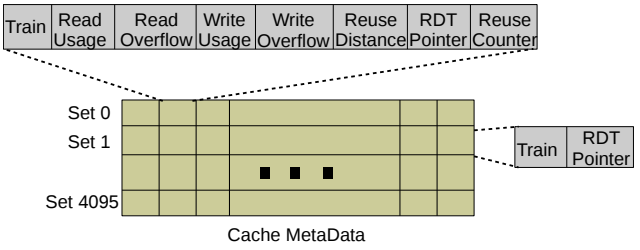


FIGURE 3.10: Organization of set sampler

be those block whose number of read-write request is predicted to have completed. Such blocks are chosen as victims. In the case of multiple candidates, blocks with short reuse category get priority first, followed by medium and then distant reuse category.

3.3.5 Set Sampling

Our predictor RDAWIP includes two additional structures: CMD and RDT. These structures introduce some area and storage overhead in the architecture (reported in the section 3.5.7). To overcome these overheads, we introduce the concept of set sampling. The main idea behind the set sampling is to use fewer sets for cache metadata. These particular sets are called as sampler sets. The use of sampler sets in the cache metadata is to train the pattern in reuse distance table. On the other hand, the other sets of the CMD are used for the decisions of prediction and replacement. Fig. 3.10 shows the organization of cache metadata after the set sampling is applied. The sampled set in the cache metadata array is after every 32 sets. The entries in the sampler set are the same as the entries in the cache metadata without set sampling. Whereas, the rest of the entries in the cache metadata have the following fields: Train (1 bit) and, RDT pointer (9 bit). Note that the RDT pointer used here is for the access (read) purpose not for the update in the RDT.

Components	Parameters
Processor	2Ghz, Dual Core and Quad Core, Alpha
L1 Cache	Private, 32 KB SRAM Split I/D caches, 4-way set associative cache, 64B block, 1-cycle latency, LRU, write-back policy
L2 Cache Bank	Shared, 1MB (256 KB SRAM and 768 KB STT-RAM) or 2MB (512 KB SRAM and 1536 KB STT-RAM), 16-way set associative cache (12-way STT-RAM and 4-way SRAM), 64B block, write-back policy
Main Memory	2GB, 160 cycle Latency
Protocol	MESI CMP Directory

TABLE 3.2: System configuration

3.4 Experimental Methodology

The section illustrates the experimental methodology used to examine the proposed architecture. Note that the detail description of the simulation framework is given at appendix A.

3.4.1 Simulator Setup

We evaluate our proposed architecture on full system simulator GEM5 [118]. Table 3.2 shows the system parameters of the evaluated system. We conducted our experiments on a dual and quad-core system with the different configurations of L2 cache. For the dual-core, we use 4 MB 16-way set associative L2 cache, and for quad-core, we use 8 MB 16-way set associative L2 cache. Besides, we also present the iso-area analysis with different sizes of baseline SRAM and STT-RAM.

Table 3.3 shows the timing and energy parameters for the different configurations of L2 cache at 32 nm technology modeled using CACTI 6.5 [19] and NVSIM [21]. We also model the energy consumption of RDAWIP by using NVSIM[2].

We compared our proposed hybrid cache architecture against the two existing techniques Read Write Aware Hybrid Cache Architecture (RWHCA) [27, 28] and Write Intensity (WI) [30] and, baseline SRAM, STT-RAM, and Hybrid Cache that uses LRU as a replacement policy with no prediction and data allocation. In RWHCA, a 2-bit counter is used to capture the access to a block. The migration

[2]The energy values of CMD are depended upon the memory size. However, the scaling of the values can be easily controlled by using sampler

Core	Cache/ Peripheral	Static Power (mW)	Read Energy (nJ)	Write Energy (nJ)	Read Latency (ns)	Write Latency (ns)
Dual Core (4MB)	SRAM	554.82	0.116	0.116	2.117	2.117
	STT	120.76	0.122	2.043	2.96	12.85
	Hybrid	229.28	0.116 / 0.122	0.116 / 2.043	2.11 / 2.96	2.11 / 12.85
	CMD	5.48	0.003	0.003	0.93	0.93
Quad Core (8 MB)	SRAM	1128.92	0.285	0.285	2.33	2.33
	STT	224.8	0.149	2.084	3.10	12.87
	Hybrid	450.83	0.285 / 0.149	0.285 / 2.08	2.33 / 3.10	2.33 / 12.87
	CMD	10.77	0.005	0.005	1.23	1.23
Octa Core (16 MB)	SRAM	2217.42	0.347	0.347	2.577	2.577
	STT	378.7	0.181	2.113	6.12	15.89
	Hybrid	838.3	0.347 / 0.180	0.347 / 2.113	2.577 / 6.12	2.577 / 15.9
	CMD	21.34	0.011	0.011	1.80	1.80
RDAWIP (CMD+RDT)	RDT	5.27	0.001	0.001	0.144	0.144
Iso Area Analysis						
Dual Core	SRAM (2MB)	287.13	0.133	0.133	1.67	1.67
	STT (6 MB)	172.78	0.135	2.063	3.03	12.86
Quad Core	STT (12 MB)	301.73	0.164	2.098	4.60	14.38

TABLE 3.3: Timing and energy parameters of the L2 cache and RDAWIP

process is triggered only when there are two consecutive accesses in the wrong region of the hybrid cache. In WI, the predictor is composed of 1024 entries that comprise a valid bit, 10-bit hashed address field (used for indexing the predictor) and a 3-bit state field. The write intensity threshold for placing the loaded block in the SRAM region is 4. In addition, each entry of the L2 cache block consists of 10-bit trigger instruction and the 2-bit counter that count the number of accesses.

The reuse distance table of our architecture consists of 512 entries. In addition, we have cache metadata that is associated with each entry of L2 cache. As we stated earlier, each block in the L2 cache is categorized into three categories: short, medium, and distant according to their reuse distance. The group of blocks that fall in the short category have reuse distance between 2^0 to 2^4. Such kind of blocks have a very short lifetime and face an extra number of accesses as compared to other blocks. Blocks fall in the group of medium reuse distance if their reuse distance is between $2^4 + 1$ to 2^6. The distant group of the block corresponds to the blocks whose reuse distance is more than 2^6. In our scheme, during the write redirection of the block to SRAM, the tag needs to be transferred from STT-RAM to SRAM. This requires an additional buffer and latency of 3 cycles (1 cycle to transfer the tag in swap buffer, one cycle for writing the tag in swap buffer and

one cycle to transfer the tag from swap buffer to SRAM region).

We also calculate the energy consumption of the additional circuits (reported in table 3.3) with the help of NVSIM. Note that the latency of RDAWIP is not on the critical path. This is mainly due to two reasons: First, the fields of the cache metadata and reuse distance table are updated simultaneously with the cache access. Second, the access of the RDAWIP happens only on the first write back from L1 cache or at the time of the eviction of the block.

3.4.2 Workloads

To evaluate our proposed HCA, we use 24 benchmarks from SPEC CPU2006 [7] benchmark suite with *ref* input. We compose twelve multi-programmed workloads for dual-core and nine multi-programmed workloads for quad-core. Table A.5 lists these multi-programmed workloads. The list is sorted according to WBKI (Write back per Kilo Instruction). In particular, for dual-core, the first four workloads (Mix1 to Mix4) show high WBKI (High1 to High4), next four workloads (Mix5 to Mix8) show average WBKI (Mid1 to Mid4) and last four workloads (Mix9 to Mix12) show low WBKI (Low1 to Low4). Whereas, in quad-core, the Mix1 to Mix3 show high WBKI (High1 to High3), Mix4 to Mix6 show average WBKI (Mid1 to Mid3) and Mix7 to Mix8 shows low WBKI (Low1 to Low9). In addition to the multi-programmed workloads, we also use five multi-threaded applications (canneal, fluidanimate, freqmine, swaptions and, x264) from the PARSEC [6] benchmark suite with *medium* input set. We briefly discuss the result analysis with multi-threaded applications in subsection 3.5.9.

3.5 Results and Analysis

The results are presented on different metrics: the total number of writes in the cache, Energy savings, Miss Per Kilo Instruction (MPKI) and Speed-up. We have conducted different simulations on different variations of the proposed technique.

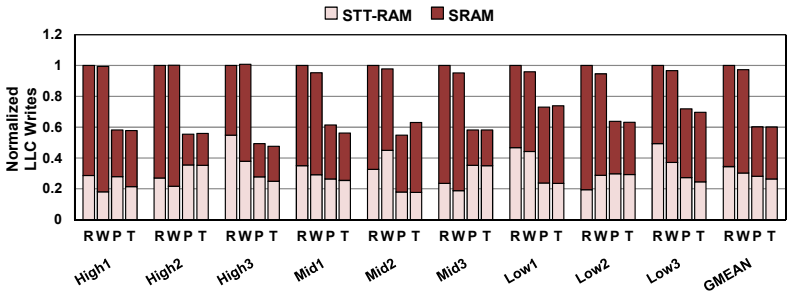


FIGURE 3.11: Normalized LLC writes of RWHCA(R), WI (W), P and T for quad core (Lower is better)

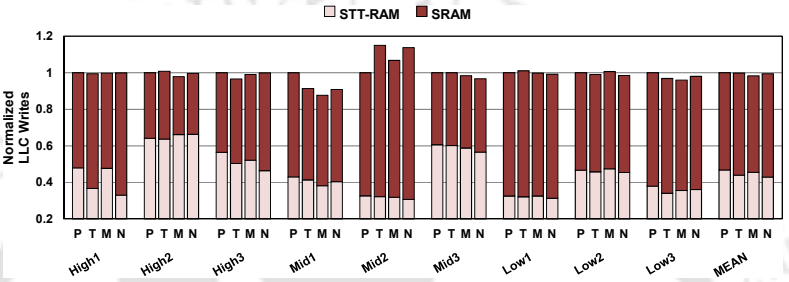


FIGURE 3.12: Normalized LLC writes of P, T, M and N for quad core (lower is better)

These variations include policy with changes in MESI protocol only (denoted by **P**), a policy with changes in MESI protocol along with the prediction of the first write back (indicated by **T**), a policy with changes in MESI protocol and replacement policy (denoted by **M**) and policy with changes in MESI protocol along with the prediction of first write back and replacement policy (indicated by **N**). Note that the strategies mentioned above are without set sampling. The results for the procedure with set sampling **(S)** is given in subsection 3.5.8. Note that the results graphs are shown for quad-core.

3.5.1 Write Accesses

Figs. 3.11 and 3.12 present the normalized write counts with respect to RWHCA (R) and P for quad-core. Table 3.4 shows the savings in write accesses with respect to RWHCA, WI, P and baselines. Note that the negative values in the table (row 8, 9, 11, 24 and 27 of table 3.4) implies the increase in writes. From the results, we conclude that the savings in writes in the STT region (row 1, 4, 7, 10, 15, 17,

Core	Policy	Region	P	T	M	N	Row
Dual	RWHCA	STT	41.9%	46.4%	43%	48.1%	1
		SRAM	50.6%	47.6%	45.1%	42.9%	2
		Total	41%	41%	40.6%	41.7%	3
	WI	STT	27.4%	33%	28.7%	35.1%	4
		SRAM	54.6%	51.9%	49.6%	47.5%	5
		Total	38.7%	38.7%	38.2%	39.4%	6
	P	STT	-	7.6%	1.8%	10.6%	7
		SRAM	-	-5.9%	-11%	-15.4%	8
		Total	-	0.02%	-0.7%	1.1%	9
	Hybrid	STT	68.5%	70.9%	69.1%	71.8%	10
		SRAM	-32.4%	-40.2%	-46.9%	-52.8%	11
		Total	33.7%	33.8%	33.2%	34.5%	12
	SRAM	SRAM	67%	65.1%	63.4%	61.9%	13
		Total	33.7%	33.7%	33.2%	34.5%	14
	STT	STT	76.7%	78.5%	77.1%	79.2%	15
		Total	34.8%	34.8%	34.3%	35.6%	16
Quad	RWHCA	STT	17.5%	22.8%	20%	25%	17
		SRAM	51%	48.6%	51.4%	48.1%	18
		Total	39.7%	39.8%	40.7%	40.1%	19
	WI	STT	7%	13%	9.8%	15.3%	20
		SRAM	52.2%	50%	52.7%	49.5%	21
		Total	38.1%	38.2%	39.1%	38.4%	22
	P	STT	-	6.3%	3%	9%	23
		SRAM	-	-4.6%	1.2%	-5.6%	24
		Total	-	0.12%	1.6%	0.52%	25
	Hybrid	STT	60.7%	63.2%	61.9%	64.2%	26
		SRAM	-35.3%	-41.5%	-33.7%	-42.8%	27
		Total	35.3%	35.4%	36.4%	35.6%	28
	SRAM	SRAM	67.1%	65.6%	67.5%	65.2%	29
		Total	36.6%	36.7%	37.6%	37%	30
	STT	STT	71.4%	73.2%	72.2%	74%	31
		Total	37.3%	37.4%	38.3%	37.6%	32

TABLE 3.4: Percentage savings in write accesses P: dataless T: dataless with prediction M: dataless with replacement N: dataless with prediction and replacement (higher is better)

20, 23, 26, and 31) is mainly due to three reasons: First: identification of private blocks and providing dataless entries to such blocks. Second: prediction of first write back from the L1 cache to dataless block in the L2 cache. Third: the large number of write-backs in the SRAM region due to the eviction of dead blocks according to their reuse distance. We also save the writes in the SRAM region of hybrid cache (row 2, 5, 13, 18, 21, and 29). This is because our policy allocates all the block loaded from the main memory in the STT region of the cache whereas the existing policies loaded the blocks in both SRAM and STT region. As we have 3:1 ratio in the hybrid cache of STT vs. SRAM, savings of writes in STT and SRAM region leads to the savings in total writes (row 3, 6, 12, 14, 16, 19, 22, 25, 28, 30, and 32). However, some of the rows (8, 9, 11, 24, and 27) of the table indicate the increase in the writes in the SRAM region. This is mainly due to the prediction of the first write back, proposed replacement policy and no data allocation/migration support in case of baseline hybrid policy.

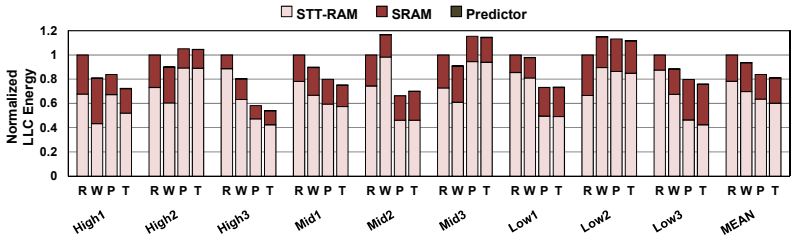


FIGURE 3.13: Normalized LLC energy of RWHCA (R), WI (W), P and T for quad core (lower is better)

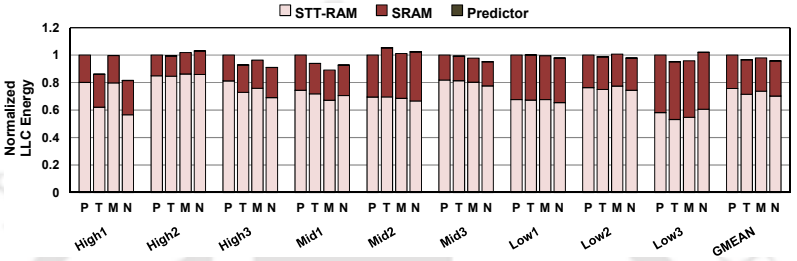


FIGURE 3.14: Normalized LLC energy of P, T, M and N for quad core (lower is better)

3.5.2 Energy Consumption

Figs. 3.13 and 3.14 show the normalized LLC energy consumption[3] for quad-core. Results with different variations of proposed technique against the existing techniques are presented in table 3.5. Note that the rows (row 2, 8, 11, 20, and 23 of table 3.5) with the negative value represents the consumption of energy. However, overall there is an improvement in total energy consumption mainly because of less number of write operations in the STT region (row 1, 4, 7, 10, 13, 16, 19, and 22 of table 3.5) of the hybrid cache. Note that we have also considered the energy consumption by the additional circuits: CMD and RDT in the overall consumption of the table 3.5 and in figs. 3.13 and 3.14.

- **RWHCA:** Compared to RWHCA, in the SRAM region, our proposed technique consumes energy (in case of dual-core) or saves less energy (in case of quad-core) (row 2 and 14 of table 3.5). This is due to the transfer of read and write-intensive blocks to the SRAM region, which in turn increases the number of hits.

[3]The normalized values are calculated from the actual energy values which are as follows (for row 1): 14.4 mJ(R), 9.2 mJ(P), 8.6 mJ(T), 9 mJ(M) and 8.3 mJ(N).

Core	Policy	Region	P	T	M	N	Row
Dual	RWHCA	STT	36.1%	40.4%	37.6%	41.8%	1
		SRAM	-1%	-9.6%	-9.6%	-14.6%	2
		Total	29.2%	32.7%	30.6%	34.3%	3
	WI	STT	20%	25.3%	21.8%	27.1%	4
		SRAM	27.2%	21%	21%	17.3%	5
		Total	17.3%	21.4%	19%	23.3%	6
	P	STT	-	6.7%	2.3%	9%	7
		SRAM	-	-8.5%	-8.6%	-13.5%	8
		Total	-	5%	2%	7.2%	9
	Hybrid	STT	59.7%	62.4%	60.7%	63.3%	10
		SRAM	-20.5%	-30.8%	-30.8%	-36.8%	11
		Total	52.9%	55.2%	53.9%	56.3%	12
Quad	RWHCA	STT	18.5%	23.1%	20.7%	24.3%	13
		SRAM	6.1%	3.5%	6.6%	1%	14
		Total	16.1%	19%	17.8%	19.6%	15
	WI	STT	8.5%	13.7%	11%	15.1%	16
		SRAM	14.2%	11.9%	14.7%	9.6%	17
		Total	10.3%	13.4%	12.2%	14.1%	18
	P	STT	-	5.6%	2.6%	7.1%	19
		SRAM	-	-2.7%	0.56%	-5.4%	20
		Total	-	3.4%	2.1%	4.2%	21
	Hybrid	STT	55.9%	58.4%	57.1%	59.1%	22
		SRAM	-41.5%	-45.4%	-40.7%	-49.2%	23
		Total	46.3%	48.1%	47.4%	48.6%	24

TABLE 3.5: Savings in energy against the existing techniques P: dataless T: dataless with prediction M: dataless with replacement N: dataless with prediction and replacement (higher is better)

- **WI:** In the case of WI, the energy savings for SRAM region (row 5 and 17 of table 3.5) are comparable because of two reasons. First, our policy does not allocate the block in the SRAM region at the load time. Second, the former technique loads the block in the SRAM according to the write intensity of the block, which in turn loads the block that is read and write-intensive.

- **P:** With respect to P, the large energy consumption for the SRAM region (row 8 and 20 of table 3.5) by the proposed variation (T, M, and N) is due to the more number of write operations on account of prediction and replacement policy which is already reported in table 3.4 (row 8 and 24).

- **Hybrid:** The large energy consumption in the SRAM region (row 11 and 23) is due to less write operation in the SRAM region (reported in row 11 and 27 of table 3.4).

  However, more energy saving in the STT region leads to total energy saving (row 3, 6, 9, 12, 15, 18, 21, 24 of table 3.5).

Results against the baseline SRAM/STT is presented in table 3.6. The static and the total energy consumption presented in the table is against the baseline SRAM

Core	Cache	Energy	P	T	M	N	Row
Dual	SRAM, STT	Static	57.9%	58.3%	58.6%	58.8%	1
		Dynamic	57.2%	59.3%	58%	60.3%	2
		Total	57.4%	57.9%	58.2%	58.4%	3
Quad	SRAM, STT	Static	59.8%	60.2%	60.4%	60.4%	4
		Dynamic	56.8%	58.3%	57.7%	58.6%	5
		Total	59.4%	59.7%	60%	60%	6

TABLE 3.6: Savings in energy against the baseline SRAM/STT-RAM (higher is better)

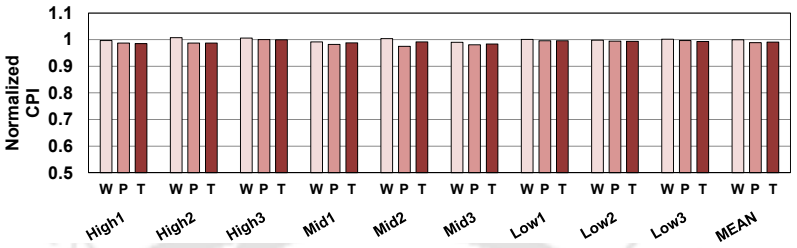


FIGURE 3.15: Normalized CPI of WI (W), P and T over RWHCA for quad core (lower is better)

(due to the large impact of static energy consumption by SRAM in total energy consumption). The improvements shown in these categories are mainly due to the low static power consumption of STT-RAM (row 1, 3, 4, and 6 of table 3.6). Whereas, the dynamic energy consumption presented in the table is against the baseline STT (due to the large dynamic energy of STT). The gain in dynamic energy (row 2 and 5 of table 3.6) is due to the less number of write operations in the STT region.

3.5.3 Performance

Fig. 3.15 presents the normalized CPI for quad-core. Our proposed techniques (P, T, M, and N) maintain the same performance with respect to RWHCA. The reason behind not getting a performance improvement despite the reduction of write accesses in the STT region of hybrid cache is because of migration or redirection of the blocks/tags to and from the SRAM region and the additional search latency for the extra states added in the directory and latency incurs due to changes in the cache controller to maintain the dataless entries. Note that the migration or redirection operation consumes extra cycles or latency.

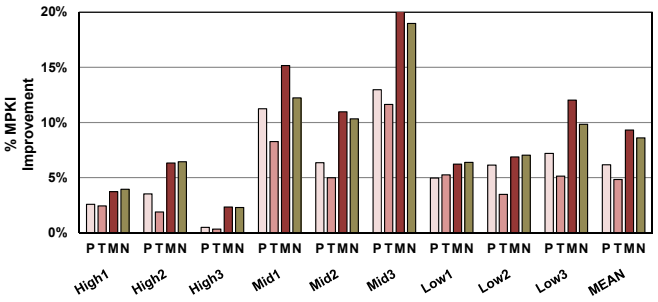


FIGURE 3.16: MPKI improvement of P, T, M and N over RWHCA for quad core (higher is better)

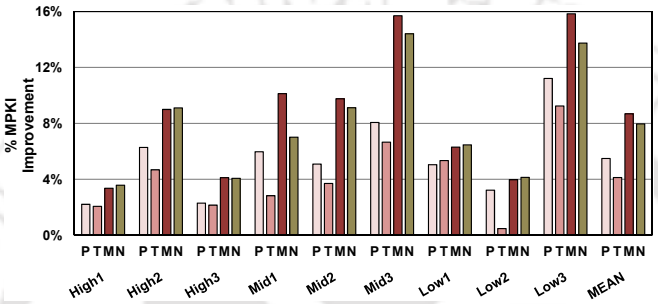


FIGURE 3.17: MPKI improvement of P, T, M and N over WI for quad core (higher is better)

3.5.4 Misses Per Kilo Instruction

Figs. 3.16 and 3.17 present the percentage improvement in MPKI over RWHCA and WI for quad-core. Table 3.7 summarizes the results obtained for the different varieties of proposed technique against the existing methods and baseline hybrid cache. As we can see from the result, the policy only with replacement (Column 5 (M)) shows the large improvement over the other proposed policy (Column 3 (P), Column 4 (T) and Column 6 (N). This is mainly due to the prioritization of some blocks for the eviction over the other blocks. On the other hand, a policy with the prediction (Column 4 (T)) shows very less improvement over the other techniques. This is due to the prediction of first write-back to limited size SRAM, which in turn increases the miss rate. Improvement in the MPKI shows that the proposed scheme reduces the main memory accesses against the existing techniques by prioritizing dead cache lines for eviction. However, the baseline Hybrid Cache (H) utilizes the full capacity without considering costly write operation. Hence, it

Core	Policy	P	T	M	N	Row
Dual	RWHCA	11.2%	10.1%	13.9%	13.4%	1
	WI	16.7%	15.6%	19.3%	18.8%	2
	P	-	-1.4%	3%	2.4%	3
	Hybrid	-18.4%	-20.2%	-14.8%	-15.5%	4
Quad	RWHCA	6.1%	4.8%	9.3%	8.6%	5
	WI	5.5%	4.1%	8.7%	7.9%	6
	P	-	-1.4%	3.4%	2.6%	7
	Hybrid	-2.5%	-4%	1.1%	0.24%	8

TABLE 3.7: MPKI Improvement (higher is better)

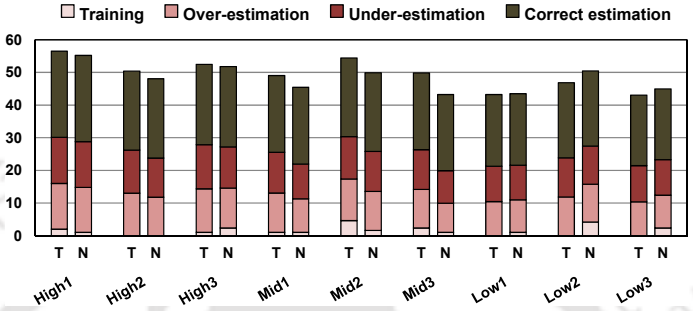


FIGURE 3.18: Accuracy of Reuse Distance Aware Write Intensity Predictor for quad core.

has the best MPKI.

3.5.5 Prediction Accuracy

Fig. 3.18 shows the accuracy of Reuse Distance Aware Write Intensity Predictor. As stated earlier in appendix A.3.3, we warm up the cache for 250 million instruction and then run each workload for 1 billion instructions. Note that the warm-up phase is required to train the predictor and the stats shown in the Fig. 3.18 are collected during the 1 billion instruction. The graph shows the log (with base 2) values of each entity. The reason behind showing the log value is due to the large values in the correct estimation. We categorize our prediction results into four categories: initialization/training, over-estimation, under-estimation, and correct estimation. These results are measured during the eviction of the block from L2 cache.

The under-estimation indicates the blocks which are read or written after they are predicted to become dead. On the other hand, over-estimation of the evicted block

Core	Metrics	Policy	P	T	M	N
Dual	Writes	SRAM 2MB	38.3%	38.4%	37.8%	39.1%
		STT 6MB	31.9%	32%	31.4%	32.7%
	EDP	SRAM 2MB	20.7%	21.8%	22.5%	22.8%
		STT 6MB	-33.5%	-31.7%	-30.4%	-30%
Quad	Writes	SRAM 4MB	39.7%	39.9%	40.8%	40.1%
		STT 12MB	34.9%	35%	35.9%	35.2%
	EDP	SRAM 4MB	20.3%	20.9%	21.5%	21.4%
		STT 12MB	-44.6%	-43.5%	-42.4%	-42.6%

TABLE 3.8: Iso area analysis

indicates that the eviction of the block happens before it becomes dead. Lastly, correct estimation refers to the evicted blocks that are correctly predicted.

From the prediction results, we find out that out of 2^{25} (approx.) evicted blocks, 2^{12} (T) and 2^{11} (N) evicted blocks are overestimated, and 2^{13} (T) and 2^{12} (N) evicted blocks are underestimated. While 2^{24} of the ejected blocks are correctly estimated. This on a whole shows the efficacy of our proposed technique.

3.5.6 Iso Area Analysis

In addition to the above-presented results, we also performed experiments on the same area footprint with baseline SRAM and STT-RAM. In particular, within the same real estate chip occupied by the proposed cache, we present analyses with the different sizes of baseline SRAM and STT-RAM. For example, within the same area of 4MB hybrid cache (i.e. 3MB STT and 1MB SRAM), we can accommodate 6MB pure STT-RAM and 2MB pure SRAM by assuming STT is 3X denser than SRAM. Table 3.8 lists the improvement in writes and EDP over the different varieties of proposed technique against pure STT and SRAM. Note that negative values in the table imply the EDP loss (row 4 and 8). The loss with respect to STT is due to the large static energy consumed by the SRAM portion of the hybrid cache. The results shown in the table 3.8 clearly indicate that within the same area footprint of SRAM, a large-sized hybrid cache can be easily accommodated.

3.5.7 Storage Overhead

Our policy uses Cache Metadata (CMD) and Reuse Distance Table (RDT) for the prediction and replacement. Each entry of CMD is associated with each cache block. The entry of CMD data comprises of many fields: Train (1 bit), Read Usage (2 bit), Read Overflow (1 bit), Write Usage (2 bit), Write Overflow (1 bit), Reuse Distance (8 bit), RDT pointer (9 bit) and the Reuse Counter (3 bit). Similarly, the entry in RDT has the following fields: Hashed Address (9 bit), Read Usage (2 bit), Read Overflow (1 bit), Write Usage (2 bit), Write Overflow (1 bit), Pointer (1 bit), Reuse Category (2 bit), and, Confidence counter (2 bit). The total number of entries in the RDT is 512. Also, a 42-bit swap buffer is used to transfer the tag from STT to SRAM. The percentage of storage overhead with respect to hybrid L2 cache is 4.90% (for dual-core) and 4.88% (for quad-core). Similarly, the percentage of area overhead with respect to hybrid L2 cache is 15% (in the case of dual-core) and 14.5% (in the case of quad-core). Note that area overhead for the additional circuit is modeled with the help of NVSIM.

3.5.8 Impact of Set Sampling

We compared the result of policy with set sampling (denoted by **S**) with the procedure without set sampling (**N**). The results are presented on the given metrics (already explained in the Section 3.5). In case of a sampler, we run each multi-programmed workload for 1 billion instructions after warming up to 1 billion instructions. The reason behind to increase the period of warm-up phase is because of the lesser availability of sampled set entries involved in initialization of the RDT. As the sampled sets are very less, the RDT will take longer time to learn the behavior of different patterns. Note that figures are only presented for quad-core.

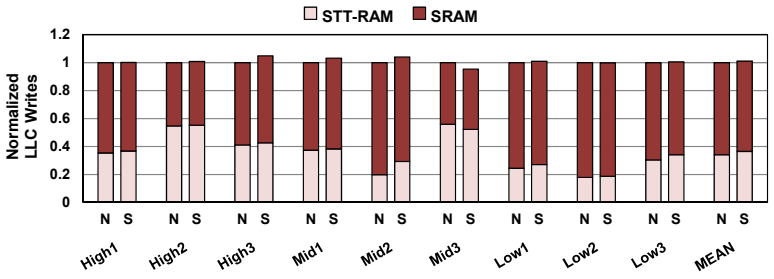


FIGURE 3.19: Normalized write counts against N.

3.5.8.1 Write Accesses

Fig. 3.19 presents the normalized LLC writes for policy N and S for the quad-core. Our policy with set sampling (S) increases the writes in STT region by 2.21% in dual-core. While, in quad-core, the increase in writes in STT is 8%. This marginal increase in writes is mainly due to slow learning of the different patterns in the RDT. Thus, the only possible way to reduce the writes in STT region in case of quad-core is by increasing the learning period. This concludes that the policy with set sampling maintains the same write accesses as the policy without set sampling.

3.5.8.2 Energy Consumption

Fig. 3.20 shows the normalized energy consumption by the policy with set sampling (S) against the policy without set sampling (N). In dual-core, the policy with set sampling (S) incurs a nominal increase in energy with respect to STT and total region by 1.31% and 0.37% respectively. Whereas, the savings in the energy in the SRAM region is only 0.80%. The marginal increase in energy of the STT region is due to the rise in the number of writes. The respective values in quad-core are the increase of 6.3% in STT region, saving of 5.5% in SRAM and, the overall rise of 1.1%. This marginal increase can be easily avoided either by increasing the sample set entries or the initialization period.

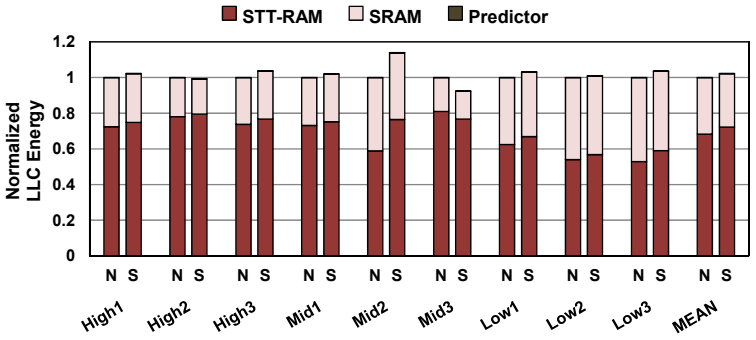


FIGURE 3.20: Normalized energy consumption against N

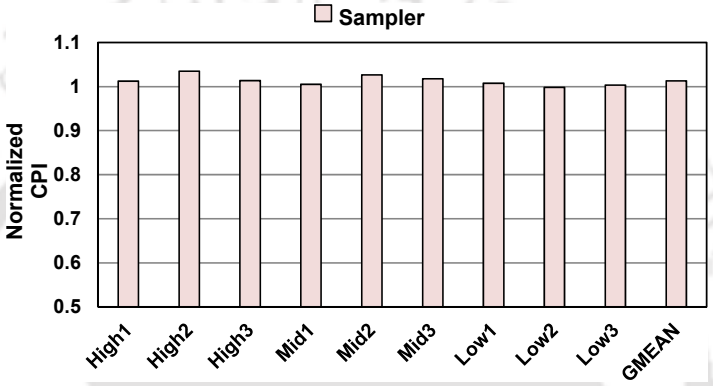


FIGURE 3.21: Normalized CPI of S against N

3.5.8.3 Performance

Normalized CPI for the policy with set sampling (S) against the policy without set sampling (N) is presented in Fig. 3.21. The policy with the sampler (S) for dual-core and quad-core maintains the same performance with the policy without sampler (N). We observe a very less degradation in CPI due to the increase of writes in STT region, but it is negligible.

3.5.8.4 Misses Per Kilo Instruction

Fig. 3.22 presents the result of an increase in MPKI of policy with set sampling (S) against the policy without set sampling (N) for quad-core. The policy with set sampling increases the MPKI by 4.25% (for dual-core) and 6.81% (for quad-core) compared to the policy without set sampling (N). This increase in MPKI is mainly due to the untrained pattern in the RDT, which replaces the block

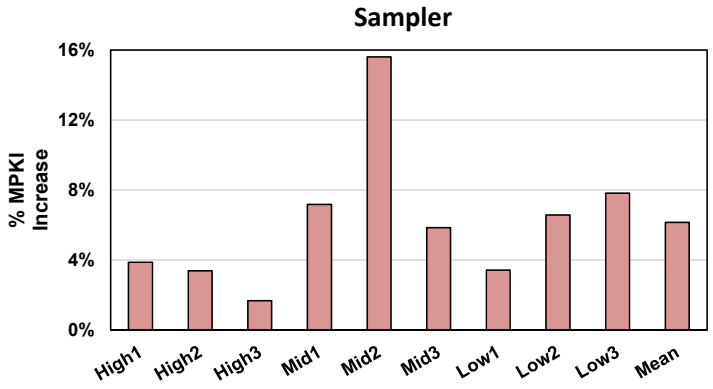


FIGURE 3.22: Increase in MPKI for S against N

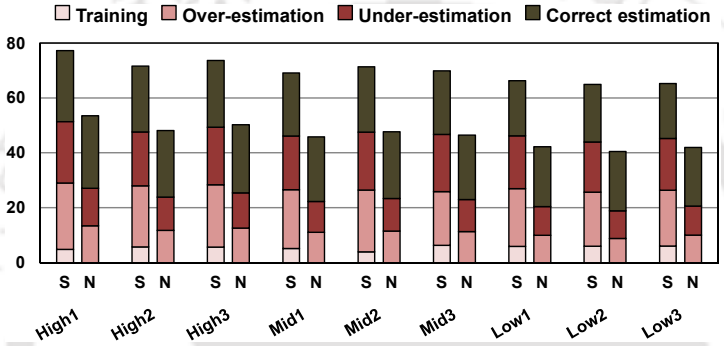


FIGURE 3.23: Accuracy of RDAWIP with set sampling for quad core

unconditionally. However, if we increase the number of sample sets entries (that leads to large storage and area overheads) or increase the warm-up time, the MPKI values can be improved further.

3.5.8.5 Prediction Accuracy

Fig. 3.23 presents the accuracy of RDAWIP with set sampling for the quad-core. Same as the previous subsection, the graph shows log (base 2) values of each entity and the stats shown in the Fig. 3.23 are collected during the 1 billion instruction after warming up the sampler for 1 billion instruction. From the graph, we can conclude that compared to policy without set sampling (N), the large number of the cache entries are going through the initialization or update phase in the policy with set sampling (S). This is because of the slow learning behavior of the RDT entry due to less availability of sampled set entries. The slow learning, in

turn, affects the prediction accuracy due to unstable entries stored in the RDT. Compared to the policy without set sampling (N), out of 2^{24} evicted blocks, 2^{19} (dual) and 2^{20} (quad) evicted blocks are underestimated and 2^{21} (dual) and 2^{22} (quad) evicted blocks are overestimated. Whereas, the 2^{23} (for dual and quad-core) evicted blocks are correctly estimated. However, if we increase the sampler size or initialization period, we can even get better prediction accuracy by trading off the area and storage overhead.

3.5.8.6 Storage Overhead

The set sampling used with the proposed policy (S) reduces the storage and area overhead compared to the policy without set sampling (N). The storage overhead with respect to hybrid L2 cache is 0.28% (for dual-core) and 0.275% (for quad-core). Similarly, the area overhead is 1.02% (for dual-core) and 0.79% (For quad-core). These less overhead are due to the limited sample set CMD entries (used to train the RDT) (which are reduced by a factor of 32 compared to policy without set sampling). Whereas, the rest of the entries in the CMD contains only Train (1 bit) and RDT pointer (9 bit).

3.5.8.7 Discussion

As it is evident from the sections 3.5.8.1, 3.5.8.2, 3.5.8.4 and 3.5.8.5, the set sampler deteriorate the metrics and prediction accuracy compared to N. However, the significant improvement in the storage and area overhead (nearly 14% in the dual and quad-core) by the use of sampling makes it worth for the embedded system having limited area.

Core	Metrics	Ref. Policy	P	T	M	N	S
Quad	Writes	RWHCA	34.4%	35.8%	34.9%	36.2%	35.9%
		WI	31.2%	32.6%	31.6%	33.1%	32.8%
		P	-	0.64%	0.42%	2.74%	2.05%
	Energy	RWHCA	32.1%	33.2%	32.4%	35%	33.5%
		WI	22.8%	24.2%	23.2%	26.1%	24.3%
		P	-	1.26%	0.47%	3.71%	1.37%
	MPKI	RWHCA	13.8%	12.1%	15.6%	15.1%	14.3%
		WI	15.3%	13.5%	17.0%	16.4%	15.7%
		P	-	-1.41%	3.57%	3.02%	2.93%
Octa	Writes	RWHCA	32.6%	33.9%	33.5%	34.8%	33.6%
		WI	26.6%	28.1%	27.6%	29%	27.7%
		P	-	2%	1.3%	3.2%	1.5%
	Energy	RWHCA	22.1%	24.4%	23.2%	25.2%	23.3%
		WI	18.1%	20.5%	19.3%	21.3%	19.3%
		P	-	1.52%	0.15%	2.6%	1.34%
	MPKI	RWHCA	1.95%	0.40%	5.93%	4.23%	4.0%
		WI	18.8%	17.4%	25.2%	23.6%	22.3%
		P	-	-2.2%	10.4%	8.65%	3.8%

TABLE 3.9: Percentage improvement values for multi-threaded workloads on quad-core and octa-core

3.5.9 Analysis on Multi-threaded Workloads with Larger Cores: Quad and Octa-core

This subsection briefly illustrates the improvements by the proposed approaches with respect to the multi-threaded applications on quad-core and octa-core. Table 3.9 presents the improvement values (in terms of Writes, Energy and MPKI) by the proposed approaches (P, T, M, N, and S) against the existing approaches (RWHCA, WI, and P have given in column 2). For performance metrics, proposed techniques maintain the same performance against the existing techniques. Note that the negative value in the table implies the increase in the value of the metric.

3.6 Summary

The key insights of this chapter are as follows:

- We presented a policy that provides dataless entries in the STT region of HCA for the blocks loaded exclusively from the main memory on an LLC miss.
- A Reuse distance aware write intensity prediction technique is proposed to predict the appropriate region to which write-backs for dataless entries should be redirected.

- In order to reduce the writing pressure in the limited sized SRAM region, we present an effective replacement policy where the victims are chosen based on the predicted lifetime of blocks. The lifetime of the block has been estimated by considering the predictor field in the replacement decision.
- Experimental results on full system simulator show that the writes are not only saved in the STT region but also in the SRAM region.
- To reduce storage and area overhead, we also propose a set sampler based prediction methodology.

To conclude, in this chapter, we presented an effective method to restrict the number of writes in the hybrid LLC. Experimental results over the two existing techniques: RWHCA and Write Intensity demonstrate a significant reduction in write accesses and energy consumption. The write accesses are reduced by 41.7% (dual-core) and 40.1% (quad-core) against RWHCA and, 39.4% (dual-core) and 38.4% (quad-core) against WI. The proposed policy also reduces energy consumption by 34.3% (dual-core) and 19.6% (quad-core) against RWHCA and, 23.3% (dual-core) and 14.1% (quad-core) against WI. By applying set sampler, the area overhead is reduced to 1.02% (dual-core) and 0.79% (quad-core). Thus, selectively placing/writing blocks in latency hungry non-volatile memories will help in effectively utilizing their potential for density and smaller leakage.

भारतीय प्रौद्योगिकी संस्थान गुवाहाटी
Indian Institute of Technology Guwahati

Chapter 4

Intra-Set Wear Leveling using Write Restricted Vertical Partitions

This chapter proposes three intra-set wear-leveling techniques: SWWR, DWWR, and DWAWR for lifetime longevity enhancement of non-volatile cache. All the strategies proposed in this chapter work on the basic concept of write restriction. Our first two techniques partition the cache into windows of equal size and distribute the writes uniformly across the cache set by employing the window as write-restricted or read-only. The selection of the write restricted window in these techniques is by rotation or by the help of counters. In our third technique, different ways of the cache are employed as a write-restricted throughout the execution to distribute the writes uniformly. The proposed works are evaluated with three different existing methods in case of dual and quad-core system.

4.1 Introduction

In the real-time execution environment, the limited write endurance of NVMs is affected by the write variations generated by the applications running on CMPs.

In cache, the write variations are categorized into two categories: *Inter-set* and *Intra-set* write variation (ref.Chapter 2). Due to these write variations, not only the lifetime of non-volatile cache reduces but the capacity of cache diminishes over the period.

Existing literature proposes many states of the art intra-set wear leveling techniques. One of them is the Probabilistic Set Line Flush (PoLF) presented by Wang et al. [29] that invalidates a cache block after a fixed number of writes based on a Flush Threshold (or FT). The only drawback here is that the possible invalidation of MRU blocks, leading to more main memory accesses. EqualChance reported by Mittal et al. [31] transfers/swaps the write-intensive blocks within the cache set with invalid/clean blocks. This transfer/swap is based on a write counter threshold Υ. However, these transfer/swaps takes extra cycles and consumes more energy due to extra writes inside a cache bank.

The main contributions of this chapter are as follows:

- Our first technique: Static Window Write Restriction (SWWR) logically divides the cache into multiple windows and using different window for write-restriction in a round-robin fashion during the execution.
- The second technique: Dynamic Window Write Restriction (DWWR) considers the write count of the different windows for the write restriction over the execution.
- In the third scheme: Dynamic Way Aware Write Restriction (DWAWR) the heavily written cache ways are considered for the write restriction during the execution.
- The presented techniques are evaluated extensively against three existing techniques: PoLF [29], WAD [32] and EqualChance [31] and the baseline STT-RAM/ReRAM without any wear-leveling support.
- Experimental evaluation over different levels of cache with different memory technologies shows significant improvement in the lifetime and reduction in intra-set write variation.

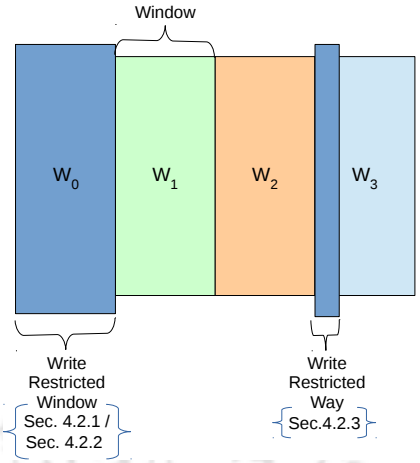


FIGURE 4.1: General overview of contribution in chapter 4

Figure 4.1 presents the general overview of the proposed contributions in this chapter.

The chapter is organized as follows: Proposed wear leveling techniques are presented in section 4.2. Section 4.3 illustrates the experimental methodology. Results and analysis are presented in section 4.4. Section 4.5 reports the parameter comparison analysis. Finally, we summarize this chapter in section 4.6.

4.2 Proposed Wear Leveling Techniques

This section demonstrates all of our proposed wear-leveling techniques: Static/Dynamic-Window Write Restriction (SWWR/DWWR) and Dynamic Way Aware Write Restriction (DWAWR). At the end of this section, we give a summary flow chart for all in fig. 4.7.

4.2.1 Static Window Write Restriction (SWWR)

4.2.1.1 Main idea

The main idea of the SWWR is to partition the cache logically into m equal-sized windows and use a different window sequentially during the execution for a specific

predefined interval (I). In each interval, one window of the cache is selected and treated as a write restricted (i.e., read-only) window. In particular, during the interval, all the writes coming from L1/L2 cache, i.e., Upper-Level Cache (called ULC) to the Write Restricted Window (WRW) of an L2/L3 cache (i.e., LLC) are redirected to other windows of the same cache set. At the end of a predefined interval, the next window of the cache is selected (as a write-restricted), and the process continues until the end of execution.

4.2.1.2 Algorithm

The working approach of the SWWR is elaborated through Algorithm 1. In our case, L2/L3 or LLC is the non-volatile STT-RAM/ReRAM based cache. In the algorithm, the tunable parameter I is used as a predefined interval (line 3). The total number of logical partitions or windows in the LLC are denoted by m (line 4). Note that each partition or window in the algorithm is represented by the variable W_i (line 5), where the range of i is from 0 to $m-1$.

When the application execution begins then for the initial I cycles, the cache is treated as an ordinarily available cache (line 6). Once the application executes I cycles, one of the windows in the cache is treated as a write-restricted window for next interval I (line 7) and, periodically for each interval, a new window is selected by rotation (line 9 to 11). The process continues until the end of execution (line 26).

When the request R comes from ULC to LLC, the tag lookup operation is performed. Depending upon the result of the lookup operation for requested **block B**, the actions are taken as given below:

- **Read Hit:** The requested block B in the LLC is served normally to higher-level cache irrespective of its location in the cache (line 13 and 14).

- **Write Hit (PUTX or write-back) and requested Block B in W_i:** If B belongs to the selected window W_i with the invalid line(s) in the other windows

Algorithm 1 Static Window Write Restriction (SWWR)

1: ULC : Upper Level Cache i.e. L1/L2.
2: LLC : Last Level Cache i.e. L2/L3.
3: I : Predefined interval.
4: m : Number of logical partitions or windows.
5: W_i : i^{th} logical partition or window that treated as read only (or write restricted) in the current interval. $0 \leq i < m$
6: Run application for I cycles treating the whole cache as normally available cache.
7: After I cycles treat one window at a time as write restricted and rotate turns in round robin fashion.
8: **repeat**
9: **for** every interval I **do**
10: $i = WINSELECT(i, m)$
11: Window W_i is selected as Write Restricted Window (WRW) for the current interval I.
12: **for** each request R from ULC to the block B in LLC during I cycles **do**
13: **if** $R = ReadHit$ **then**
14: $NORMOPR(R, B)$
15: **else if** $R = WriteHit$ **then**
16: **if** $B \in W_i$ **then**
17: $WRITEREDIRECT(R, B, \text{WRW})$
18: **else**
19: $NORMOPR(R, B)$
20: **end if**
21: **else**
22: $PROCESSCACHEMISS(R, \text{WRW})$ ▷ **cache miss**
23: **end if**
24: **end for**
25: **end for**
26: **until** the end of the execution

Write Restricted Window Selection for SWWR

27: **function** WinSelect(i, m)
28: $i = (i + 1)\%m$
29: **return** i
30: **end function**

Functions used by the Algorithms

31: **function** NormOpr(R, B)
32: Request R is served normally from block B as the conventional cache.
33: **end function**
34: **function** WriteRedirect(R, B, WRW)
35: Write the block B to other location L in the same cache set. Note that the location L does not belong to currently selected WRW.
36: **return** L
37: **end function**
38: **function** processCacheMiss(R, WRW)
39: Forward the Request R to main memory to fetch the block. Keep the newly arrived block in a location other than WRW location.
40: **end function**

of the same cache set, the request R from ULC is redirected to the first invalid line and the Block B is invalidated from the respective location in write restricted window. In the other case, when there is no invalid line in the other windows of the same cache set, the LRU victim line v is picked from the Location L and the write-back operation is performed according to its dirty bit. Note that the location L is the location other than the write restricted window location. Once the victim v is evicted from the LLC, the write request from a ULC is redirected to the generated location L and the Block B is invalidated from its respective location in WRW (line 15 to 17).

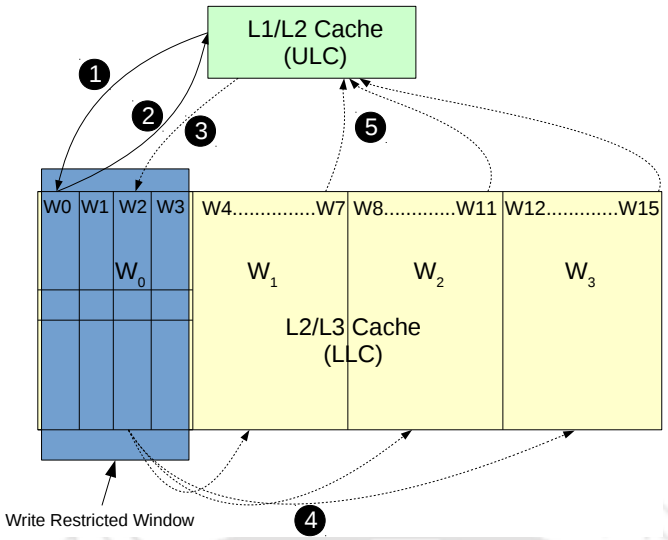


FIGURE 4.2: Working example of proposed SWWR wear leveling policy

- **Write Hit (PUTX or write-back) and B not in W_i:** The requested Block B in the LLC is written normally by the ULC (line 18 to 20).

- **LLC miss:** When the requested Block B is not present in the LLC, the request R from the ULC is forwarded to the next level of the memory hierarchy (i.e., main memory in our case). In this case, the newly arrived Block is placed in the location other than the WRW W_i location (line 21 to 23).

4.2.1.3 Working Example

Figure 4.2 depicts the working methodology of the SWWR. In the figure, 16-way set-associative LLC is partitioned into 4 ($m = 4$) equal-sized windows (W_0, W_1, W_2 and W_3). Each window of the LLC contains four ways and, the write restricted window is in W_0, i.e., way-0 to way-3. To explain the methodology, we used the arrows to show the request and response from the ULC and LLC. For the read request (shown by the arrow 1) to the way-0 of LLC, the response is served normally by the LLC (as indicated by arrow 2). On a write request from ULC cache to the block belongs to W_0 of the LLC (arrow 3), the request is redirected to the one of the free cache way(s) of W_1, W_2 and W_3. In case, if all the cache ways are occupied, the LRU victim among these ways is selected, and the write-back

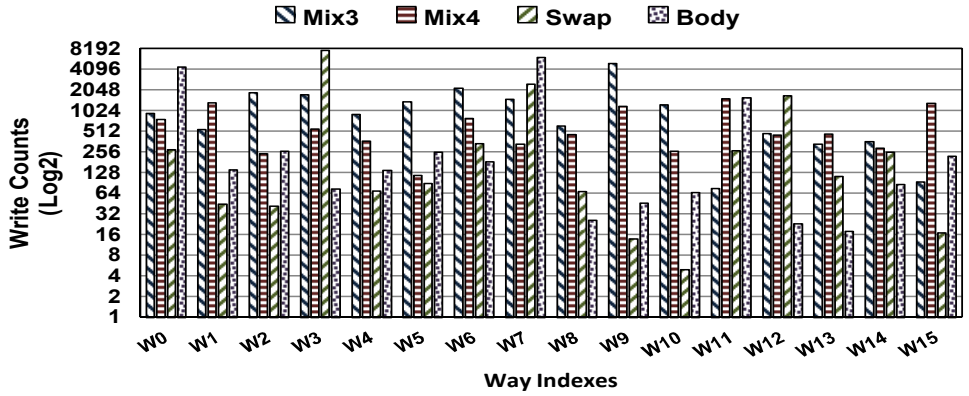


FIGURE 4.3: Write counts for four different workloads in the SWWR

Workloads		H-Win		
Type	**Bench**	**1**	**2**	**Avg.**
PARSEC	**Body**	14.4%	3.01%	17.44%
	Cann	2.54%	-	2.54%
	Ded	6.03%	1.41%	7.44%
	Swap	17.2%	7.27%	24.5%
	X264	10.04%	-	10.04%
SPEC	**Mix1**	11.17%	3.71%	14.88%
	Mix2	9.34%	4.11%	13.45%
	Mix3	9.88%	4.26%	14.14%
	Mix4	12.4%	4.74%	17.2%
MEAN		**10.35%**	**3.17%**	**13.51%**

TABLE 4.1: Percentage times Heavily written Window (H-Win) available in cache

operation is scheduled with the redirection of a write request (arrow 4). Once the write operation is performed, the write-back acknowledgment is sent to the ULC (arrow 5), and the requested block is invalidated from the W_0.

4.2.1.4 Limitation of SWWR

Figure 4.3 presents the effects in the write counts of different blocks inside the cache sets after applying SWWR. As can be seen from the figure, the maximum write count is reduced compared to write count of the Non-Volatile without any wear-leveling support as shown in the figure 2.9 (Details about the experimental setup is reported in section 4.3). However, the limitation of the SWWR is the lack of consideration of write intensity of other windows in the window selection process. Because of write variation generated by the applications, the write intensity of the windows changes over the period. In other words, during execution, the lightly

written window becomes heavily written, and vice-versa. By considering these heavily written windows as a write-restricted during the execution, we can further improve the relative lifetime and reduce the coefficient of intra-set write variation (observed in figure 4.3 for SWWR). Table 4.1 shows the percentage availability of heavily written windows (H-win) during the window selection process in SWWR. From the table, we can conclude that on an average 13.51% times a heavily written window other than the selected write restricted window is available in the cache. This motivates us to identify such windows and improve the lifetime further.

4.2.2 Dynamic Window Write Restriction (DWWR)

4.2.2.1 Main Idea

The main idea of DWWR is to partition the cache into m equal-sized windows and use different window during the execution for a predefined interval. In SWWR, the selection of write restricted window was in a round-robin fashion. Whereas, in DWWR, the selection of the window is based on a counter associated with each window. In particular, with each window, we have added a counter that is used to track the number of writes during the past interval to the window (i.e., from ULC to LLC). Then, for each interval, the window with maximum writes is chosen and treated as write restricted (or read-only) window. At the end of the interval, the next window of the cache is selected based upon the counter values, and the process continues until the end of execution. Note that, to remove the possibility of selecting the same window in the consecutive intervals, we reset the counter for the chosen write restricted window.

4.2.2.2 Algorithm

The working approach of DWWR is elaborated through Algorithm 2. In the algorithm, the partition or window of the cache is represented by the variable W_i. Similarly, the counter associated with each window is represented by the variable C_i (line 5 and 6). Note that the range of i is from 0 to $m - 1$.

Algorithm 2 Dynamic Window Write Restriction (DWWR)

1: ULC : Upper Level Cache i.e. L1/L2.
2: LLC : Last Level Cache i.e. L2/L3.
3: I : Predefined interval.
4: m : Number of logical partitions or windows.
5: W_i : i^{th} logical partition or window that is treated as read-only (or write restricted) in the current interval. $0 \leq i < m$
6: C_i : Counter corresponding to i^{th} window that records the number of write accesses from ULC to that window. $0 \leq i < m$.
7: Run application for I cycles treating the whole cache as normally available cache.
8: After I cycles treat one window at a time as read-only or write restricted.
9: **repeat**
10: **for** every interval I **do**
11: $i = WINSELECT(C_i, m)$
12: Let W_i is the selected Write Restricted Window (WRW) for current interval I.
13: **for** each request R from ULC to the block B in LLC during I cycles **do**
14: **if** $R = ReadHit$ **then**
15: $NORMOPR(R, B)$
16: **else if** $R = WriteHit$ **then**
17: **if** $B \in W_i$ **then**
18: $L = WRITEREDIRECT(R, B, \text{WRW})$
19: The corresponding counter of the window that contains the location L is incremented.
20: **else**
21: $NORMOPR(R, B)$
22: Increment the counter C_j of the window where the block B is present.
23: **end if**
24: **else**
25: $PROCESSCACHEMISS(R, \text{WRW})$ ▷ **cache miss**
26: **end if**
27: **end for**
28: **end for**
29: **until** the end of the execution

Write Restricted Window Selection for DWWR

30: **function** WinSelect(C_i, m)
31: $i = max(C_i), \quad 0 \leq i < m$
32: Reset the counter C_i to zero
33: **return** i
34: **end function**

For the initial I cycles of the application execution, the cache is employed as an ordinarily available cache (line 7). Once the application crosses the I cycles, one of the windows of the cache is treated as read-only or write restricted window. The selection of the window is based upon the counter associated with each window of the cache (line 10 and 11). Thus, the window with most write accesses in the previous intervals is selected as a write-restricted in the next interval (line 31). This helps to restrict the heavily written window to get further writes in the next interval. Once the window is selected, the corresponding counter is reset to zero (line 32). At the end of the interval, the next window with maximum writes is selected by the method. The process continues until the end of execution (line 29).

Same as in SWWR, the tag lookup operation is performed on the LLC, for each request coming from the Upper-Level cache (ULC) (line 13). Depending upon the

outcome of the lookup for requested Block B, different operations are performed in the LLC which are as follows:

- **Read Hit:** The read operation is performed in the same way as given in Algorithm 1 (line 14 and 15 of Algorithm 2).

- **Write Hit (Write-back or PUTX) and block B in W_i:** The write request for the block B in W_i is redirected to the first invalid way(s) of the same cache set in the other windows. In case, if there is no invalid line available in the same cache set, the victim line v is evicted from the location L. Note that location L is different from the write restricted window location. Afterward, the write request R from a ULC is redirected to the location L (let say in window W_j). The counter C_j corresponding to the window W_j is incremented and the block B is invalidated from W_i (line 16 to 19).

- **Write Hit (Write-back or PUTX) and block B not in W_i:** The write request R is performed normally on the block B. The corresponding counter C_j of window W_j (in which the write operation is performed) is incremented (line 20 to 22).

- **Cache Miss:** In the case of LLC miss, the request R from the ULC is forwarded to the next level of memory (main memory in our case). When the requested block has arrived at the LLC, it will be placed in the location apart from the location belonging to W_i (line 24 to 26).

4.2.2.3 Working Example

Figure 4.4 presents the working example of DWWR. As shown in the figure, the counters C_0 to C_3 are associated with the window W_0 to W_3. Let W_0 be the write restricted window having the maximum write count i.e. $C_0 = max(C_i),\ \forall i$. In the first case, a read request from the ULC to LLC (arrow 1) is served normally (arrow 2). In the second case, a write request from the ULC to the way-2 of LLC (arrow 3) is redirected to another way in the same cache set (arrow 4). In case, if

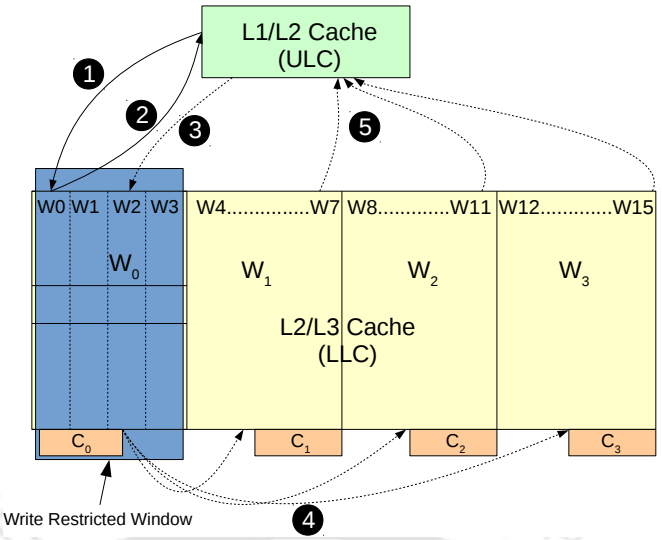


FIGURE 4.4: Working example of proposed DWWR wear leveling policy

Workloads		H-Ways								
Type	**Bench**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**Avg.**
PARSEC	**Body**	24.9%	7.4%	4.9%	4.1%	0.76%	1.16%	1.22%	0.24%	44.7%
	Cann	11.8%	8.14%	4.9%	2.35%	0.78%	0.33%	0.11%	-	28.5%
	Ded	4.85%	3.41%	2.35%	1.49%	-	-	-	-	12.1%
	Swap	23.2%	17.4%	11.6%	7.34%	2.14%	1.4%	0.61%	0.54%	64.3%
	X264	6.07%	2.82%	1.83%	1.03%	-	-	-	-	12.2%
SPEC	**Mix1**	2.78%	2.79%	1.82%	1.10%	0.73%	0.34%	0.13%	-	9.7%
	Mix2	4.14%	3.69%	3.03%	2.47%	1.46%	0.96%	0.49%	0.29%	16.57%
	Mix3	5.74%	4.07%	2.87%	1.88%	0.47%	0.38%	0.19%	0.12%	15.7%
	Mix4	7.26%	5.91%	4.42%	3.13%	1.16%	0.74%	0.39%	0.18%	23.2%
MEAN		**10.1%**	**6.18%**	**4.21%**	**2.76%**	**1.07%**	**0.76%**	**0.45%**	**0.23%**	**25.2%**

TABLE 4.2: Percentage times Heavily written Ways (H-Ways) (apart from write restricted ways) present in the cache

there is no invalid line is present in the cache way, one of the blocks is invalidated from other windows (W_1, W_2 and W_3 in our example) in the same cache set. Once the write operation is performed, the block (present in the way-2) is invalidated from the write restricted window W_0. Afterward, the corresponding counter of the window in which the write redirection happens is incremented.

4.2.2.4 Limitation of DWWR

Figure 4.5 shows the write counts of blocks within the cache set after incorporating DWWR. However, the limitation of DWWR is the lack of consideration for heavily written ways available in the other lightly written windows. In other words, a particular window may have a lower write count compared to others but can

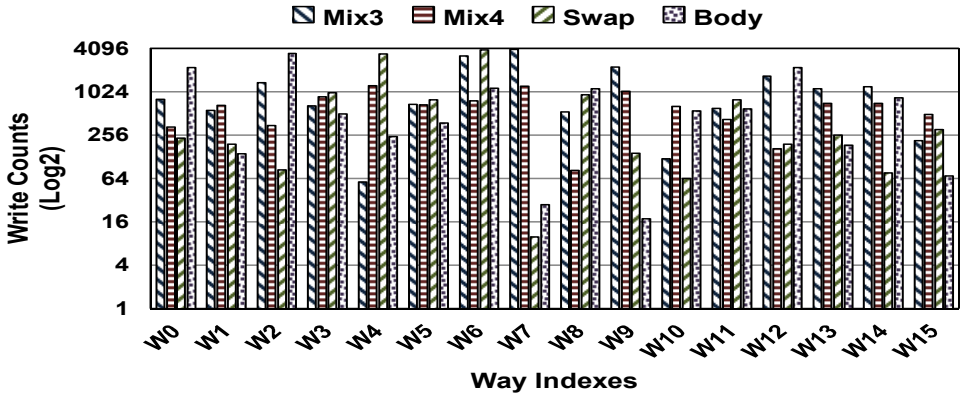


FIGURE 4.5: Write counts for different workloads in DWWR

contain some ways that are written more number of times compared to those in windows with overall higher write count. Table 4.2 presents the availability percentage of heavily written ways (H-ways) in the windows apart from the selected window. From the table 4.2, we can conclude that on an average 25.2% times H-ways are present in windows other than the selected window of the cache. This motivates us to identify such H-ways in the cache and further reduce the intra-set write variation (shown in figs. 4.3 and 4.5) and improve the lifetime over SWWR and DWWR.

4.2.3 Dynamic Way Aware Write Restriction (DWAWR)

4.2.3.1 Main Idea

The main idea of DWAWR is to select the n heavily written ways and designate them as write-restricted for a specific predefined interval I over the execution. The selection of cache ways in DWAWR is based upon examining the way counters (Z) associated with each cache way. The way counter is used to track the number of write accesses in the past intervals from the ULC to that way in LLC. In each interval, the n ways with maximum writes are selected and treated as read-only (or write restricted). At the end of the interval, the next n ways are chosen for the next interval, and the process continues until the end of execution. Note that to remove the possibility of choosing the same way(s) in the successive intervals, the

counters (Z) associated with the selected ways of the previous interval are reset to zero.

4.2.3.2 Algorithm

Algorithm 3 Dynamic Way Aware Write Restriction (DWAWR)

1: ULC : Upper Level Cache i.e. L1/L2.
2: LLC : Last Level Cache i.e. L2/L3.
3: I : Predefined interval.
4: n : Number of ways that is treated as read only (or write restricted).
5: List $< integer > waysList$: List of Write Restricted Ways (WRW) in the current interval. Size of list is n.
6: Z_j : Way counter with respect to j^{th} way that records the number of write accesses from L1/L2 cache to the way. $0 \leq j < cache\_assoc$.
7: Run application for I cycles treating the whole cache as normally available cache.
8: After I cycles treat n ways as read-only or write restricted.
9: **repeat**
10: **for** every interval I **do**
11: $WINSELECT(Z_j, n, waysList)$
12: **for** each request R from L1/L2 cache to the block B in L2/L3 cache during I cycles **do**
13: **if** $R = ReadHit$ **then**
14: $NORMOPR(R, B)$
15: **else if** $R = WriteHit$ **then**
16: **if** $B \in waysList$ **then**
17: $L = WRITEREDIRECT(R, B, \text{WRW})$
18: The corresponding counter Z_L of the location L is incremented.
19: **else**
20: $NORMOPR(R, B)$
21: Increment the counter Z_j of the cache way where the block B is present.
22: **end if**
23: **else**
24: $PROCESSCACHEMISS(R, \text{WRW})$ ▷ **cache miss**
25: **end if**
26: **end for**
27: **end for**
28: **until** the end of the execution

Write Restricted Way Selection for DWAWR

29: **function** WinSelect(Z_j, n, $waysList$)
30: **for** $k \leftarrow 0$ to n **do**
31: Let Z_j be the maximum counter in the cache. $0 \leq j < cache\_assoc$
32: $waysList.add(j)$, $0 \leq j < cache\_assoc$
33: $Z_j = 0$
34: **end for**
35: **end function**

Algorithm 3 presents the working approach of DWAWR. In the algorithm, the use of predefined interval (I) is same as the SWWR and DWWR (line 3). The parameter n acts here as the number of ways that are treated as write-restricted (line 4). $waysList$ is the list of integers of size n. It contains the list of ways in the cache that are treated as write-restricted in the current interval (line 5). The way counter of the cache is represented by Z_j (line 6). Note that the range of j is from 0 to $A - 1$ (where A is the cache associativity).

Same as in the previous approaches, for the initial I cycles of application execution, the cache is employed as an ordinarily available cache (line 7). Once the application crosses the I cycles, the n ways of the cache with maximum write counts are selected and treated as write-restricted way (or read-only) for the next interval, I (line 8). In particular, the way counters having maximum value in the previous interval is selected in the next interval (line 11, 30 to 32). Once the ways are selected for the write restriction, the counters corresponding to the ways are reset to zero (line 33). This restricts the chances of heavily written ways in the previous intervals to get further more writes in the next interval. At the end of the interval, the next n ways are selected, and the process continues until the end of execution (line 28).

For each request coming from ULC to LLC, the tag lookup operation is performed on the cache. Depending upon the result of the lookup operation, the read (line 13 and 14), write (line 15 to 22) and forward to main memory operation (line 23 to 25) is performed in the LLC. The handling of these operations is already described in section 4.2.2.2. The only difference is that in case of the write operation, the corresponding counter of the way in which the write is redirected or in which the write operation is performed is incremented.

4.2.3.3 Working Example

Figure 4.6 depicts the working example of the proposed DWAWR approach. In the example, the way counters Z_0 to Z_{15} are associated with each way W_0 to W_{15} of the cache. Let way-0, way-5, way-12, and way-15 are selected as write-restricted ways in the current interval. In the first case, the read request from the ULC to LLC (arrow 1) is served normally (arrow 2) by the LLC. In the second case, the write request from ULC coming to way-5 of LLC (arrow 3) is redirected (arrow 4) to one of the inferior ways of the cache (let say way-2, way-8, and way-13) depending upon the availability of invalid and victim location in the cache. Once the write operation is performed, the write-back acknowledgment is sent back to the ULC (arrow-5).

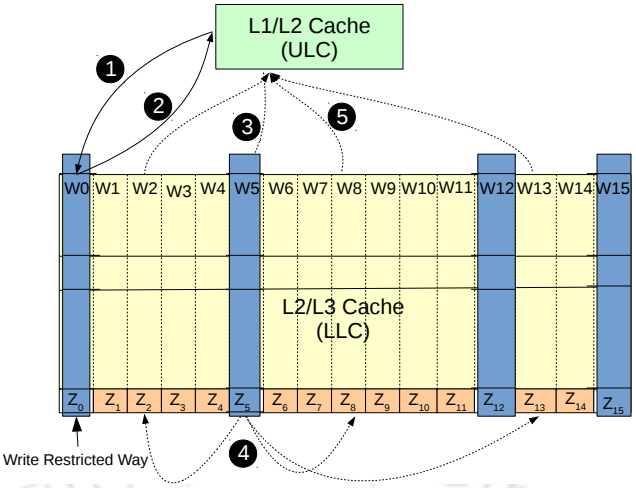


FIGURE 4.6: Working example of proposed DWAWR wear leveling policy

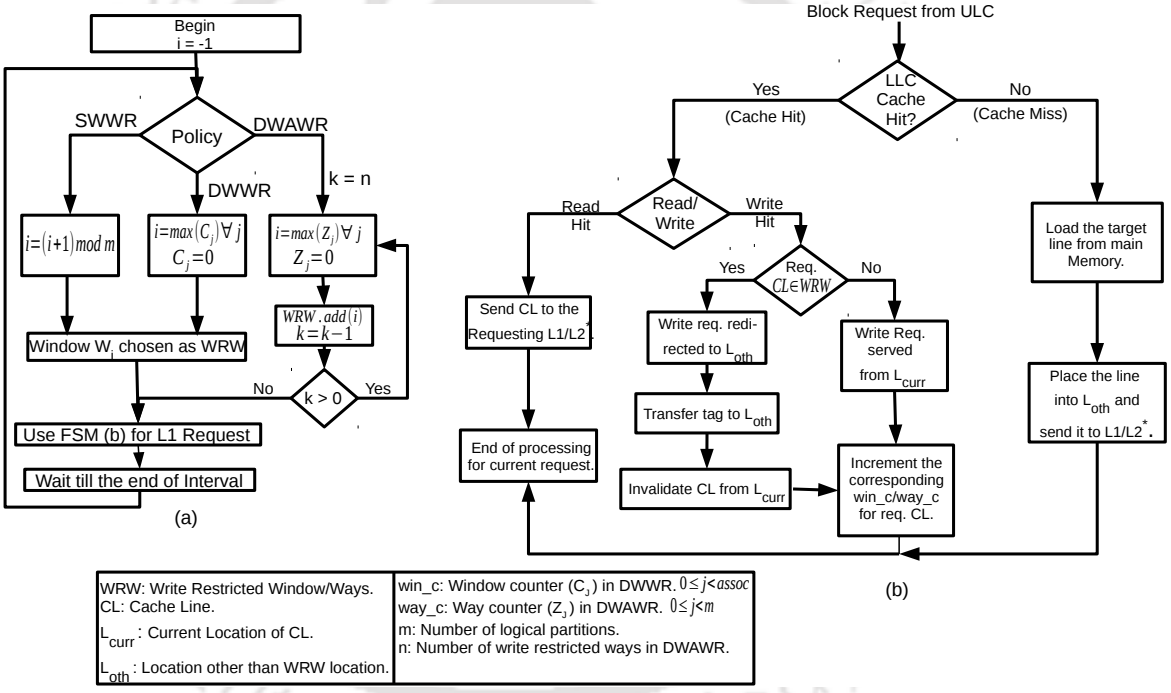


FIGURE 4.7: (a) WRW construction flow chart (b)The working flow chart summarizing the proposed schemes: SWWR, DWWR and DWAWR

The working flow diagram of the presented approaches: SWWR, DWWR and DWAWR is summarized in fig. 4.7.

Components	Parameters
Processor	2Ghz, Dual Core and Quad Core, X86
L1 Cache	Private, 32 KB SRAM Split I/D caches, 4-way set-associative cache, 64B block, 1-cycle latency, LRU, write-back policy
Two-Level Cache Memory Parameters	
L2 Cache	Shared, STT-RAM, 64B block, LRU, write-back policy
Three-Level Cache Memory Parameters	
L2 Cache	Private, 512 KB SRAM, 8-way set-associative cache, 64B block, 5-cycle latency, LRU, write-back policy
L3 Cache	Shared, ReRAM, 16-way set-associative cache, 64B block, LRU, write-back policy
Main Memory	2GB, 160 cycle Latency
Protocol	MESI CMP Directory

TABLE 4.3: System parameters

4.3 Experimental Methodology

The section describes the experimental methodology employed to evaluate the proposed architectures: SWWR, DWWR, and DWAWR. For the detail description of the simulation framework, refer appendix A.

4.3.1 Simulator Setup

We evaluate our proposed approaches on a full system simulator GEM-5 [118]. Table 4.3 shows the system parameters used in the simulation. We conducted our experiments on a dual and quad-core system with different levels of cache made up of different memory technologies and with distinct configurations of caches and parameters. Table 4.4 shows the timing and energy parameter for these configurations obtained by using NVSIM [21] at 32nm technology node. We employed STT-RAM as an LLC in two-level cache hierarchy and ReRAM as an LLC in a three-level cache hierarchy. The reasons behind to choose the STT-RAM as the LLC in the 2-level cache hierarchy is because the LLC at two-level cache hierarchy experience more number of writes as compared to LLC at 3-level cache hierarchy. With large write endurance, the STT-RAM is best suited for level-2 LLC compared to ReRAM.

We compared our proposed approaches against three existing approaches: Probabilistic Set Line Flush (PoLF) [29], Write-Back Aware Intraset Displacement

LLC Configuration	Leakage Power (mW)	Hit Energy (nJ)	Miss Energy (nJ)	Write Energy (nJ)	Hit Latency (ns)	Miss Latency (ns)	Write Latency (ns)
Two Level STT-RAM L2 Cache System							
32MB, 16way	17.187	0.544	0.093	4.261	259.536	16.8	747.147
16MB, 32way	15.668	0.457	0.186	6.548	84.074	17.475	271.035
16MB, 16way	15.674	0.367	0.096	4.322	78.453	11.854	271.035
16MB, 8way	15.659	0.317	0.047	3.244	78.283	11.684	271.035
8MB, 32way	8.116	0.366	0.185	6.454	74.792	8.259	270.981
8MB, 16way	8.030	0.273	0.093	4.387	78.497	11.964	270.981
8MB, 8way	7.983	0.227	0.047	3.221	74.454	7.921	270.981
4MB, 16way	7.960	0.217	0.093	4.228	23.876	5.575	126.585
Three Level ReRAM L3 Cache System							
8MB, 16way	60.196	0.65	0.093	1.62	54.71	48.66	67.71
16MB, 16way	132.32	1.128	0.122	2.078	54.92	48.702	67.736

TABLE 4.4: Timing and energy parameters for STT-RAM/ReRAM LLC

(WAD) [32] and EqualChance [31] and, the baseline STT-RAM/ReRAM that uses LRU as a replacement policy with no wear leveling policy associated. In PoLF, the value of Flush Threshold or FT for skipping the write operation is set to 16. In WAD, we set the value of the intraset saturation counter (RSC) to 7 and use the clean-LRU block intraset displacement approach. On the other hand, in EqualChance, a 4-bit write counter along with a flag bit are associated with each set. The value of Υ to trigger the transfer/swap operation within the cache set is set to 16. In addition to the write counter, a 64B swap buffer is used for the swap operation of the block within the cache set. Note that we model only the single swap buffer in our setup as the bank contention model is used in our approaches. The C-shifting and I-shifting in Equalchance take extra cycles for transfer/swap operation. We model these extra cycle in our simulator setup as same as in [31].

In SWWR, DWWR, and DWAWR, during the write redirection, the tag needs to be transferred from the current write restricted window/way to the new location within the cache set. The transfer of tag requires an additional 42-bit buffer and additional latency of 3 cycles (one cycle to transfer the tag in tag buffer, one cycle to writing the tag in tag buffer and, one cycle to transfer the tag to the new location within the cache set). In DWWR, the size of the counter C_i associated with each window is set to 13 bit. Whereas, in DWAWR, the size of the way counter Z_j is set to 11 bit. Later, we analyze the appropriate sizes of the counter values in the section 4.5.5 with different cache configurations and parameters. Note that the selection of window/way in DWWR and DWAWR is not on the critical path.

Benchmark Suite	Benchmarks			
PARSEC v2.1	Bodytrack (Body), Canneal (Cann), Dedup, Swaptions (Swap), X264			
SPEC CPU2006	**Dual Core Workloads**		**Quad Core Workloads**	
	Mix1	bwaves, gamess	Mix1	zeusmp, bwaves, leslie3d, cactusADM
	Mix2	lbm, zeusmp		
	Mix3	perlbench, bzip2	Mix2	perlbench, omnetpp, gcc, libquantum
	Mix4	gcc, bzip2		
	Mix5	omnetpp, milc	Mix3	gobmk, tonto, sjeng, namd
	Mix6	dealII, namd		
	Mix7	h264ref, gobmk	Mix4	calculix, astar, dealII, h264ref
	Mix8	sjeng, calculix		

TABLE 4.5: Benchmarks used for evaluation

This is because, in the proposed approaches, the windows/way selection happens at the end of the interval before the arrival of a new request.

4.3.2 Workloads

We verified our proposed approaches on both multi-threaded: PARSEC [6] and multi-programmed: SPEC CPU 2006 [7] benchmark suites. Five benchmarks with *medium* input set are used from the PARSEC. Twenty benchmarks with *ref* input are used from SPEC CPU 2006. Table 4.5 lists the name of the benchmarks used for the evaluation and the mixes of applications for the multi-programmed workload.

4.4 Results and Analysis

4.4.1 Two Level Cache Analysis: L2-STT-RAM

We evaluate our proposed approaches: SWWR, DWWR, and DWAWR on the dual and quad-core system. For a dual-core system, an 8MB 16-way set-associative STT-RAM L2 cache is used and, for the quad-core system, 16MB 16-way set-associative STT-RAM L2 cache is used. In our evaluation, we set the values of I (predefined interval) to 2 million cycles in case of dual-core and 1 million cycles in case of quad-core and, m (SWWR/DWWR)/n (DWAWR) to 4. The rationale behind the different interval values is that the number of accesses is doubled in

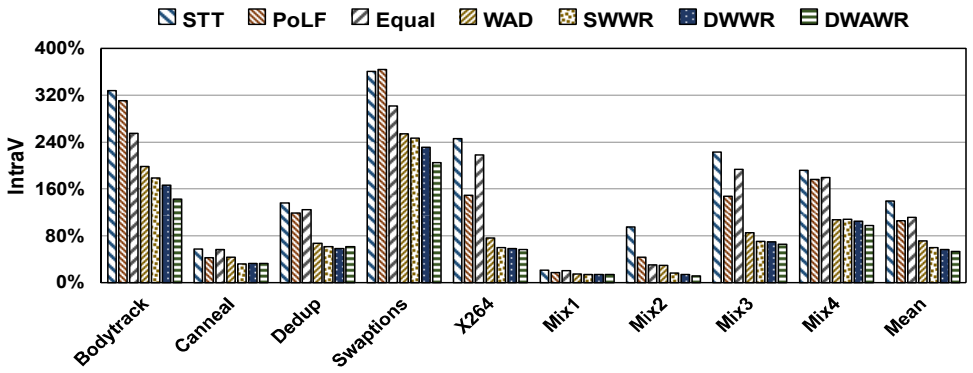


FIGURE 4.8: Intra-Set write variation for quad core (lower is better)

case of a quad-core system compared to the dual-core. Later in the section 4.5, we analyze the effects of changing these values. Note that the graphs are only given for quad-core.

We present our results on the following metrics: coefficient of Intra-set write variation ($IntraV$) calculated with the help of Equation (2.2), coefficient of Inter-set write variation ($InterV$) calculated by using Equation (2.1), relative lifetime improvement calculated by using Equation (2.3), energy overhead, speedup and number of invalidations/flushes.

4.4.1.1 Coefficient of Intra-Set Write Variation

Figure 4.8 presents the coefficient of intra-set write variation for quad-core system. Table 4.6 lists the percentage reduction in the coefficient of intra-set write variation by the proposed approaches against the existing techniques and, the baseline STT-RAM for both multi-threaded and multi-programmed applications. Note that the negative values (row 14) in the table implies the increase of write variation.

- **STT:** Compared to baseline STT, the reduction in intra-set write variation (row 1-3 and 16-18) is mainly due to the more effective uniform write distribution by the proposed approaches.

Core	Reference Policy	Suites	SWWR	DWWR	DWAWR	Row
Dual	STT	PARSEC	94.2%	95.4%	101.6%	1
		SPEC	41%	40.8%	42%	2
		Total	56.7%	56.9%	59.1%	3
	PoLF	PARSEC	43.3%	44.6%	50.8%	4
		SPEC	19%	18.8%	20%	5
		Total	26.2%	26.4%	28.6%	6
	Equal	PARSEC	38.7%	40%	46.2%	7
		SPEC	31.2%	31%	32.2%	8
		Total	35.4%	35.6%	37.9%	9
	WAD	PARSEC	11.4%	12.7%	18.9%	10
		SPEC	12.1%	11.9%	13%	11
		Total	13.1%	13.3%	15.5%	12
	SWWR	PARSEC	-	1.3%	7.5%	13
		SPEC	-	-0.20%	1%	14
		Total	-	0.2%	2.4%	15
Quad	STT	PARSEC	99.2%	102.7%	106.7%	16
		SPEC	60.2%	62.4%	65.2%	17
		Total	80%	82.9%	86.5%	18
	PoLF	PARSEC	65.8%	69.3%	73.3%	19
		SPEC	29.4%	31.6%	34.4%	20
		Total	46%	48.9%	52.5%	21
	Equal	PARSEC	76.2%	79.7%	83.7%	22
		SPEC	31.9%	34.1%	36.9%	23
		Total	51.7%	54.6%	58.2%	24
	WAD	PARSEC	14.6%	18.2%	22.2%	25
		SPEC	8.4%	10.6%	13.4%	26
		Total	11.4%	14.3%	17.9%	27
	SWWR	PARSEC	-	3.6%	7.6%	28
		SPEC	-	2.2%	5%	29
		Total	-	2.9%	6.5%	30

TABLE 4.6: Percentage reduction in coefficient of Intra-Set write variation (higher is better)

• **PoLF:** The improvement in the write variation (row 4-6 and 19-21) over PoLF is because the existing method invalidates the data block randomly or probabilistically without concerning its write behavior. On the other hand, our proposed technique SWWR redirects the heavily written block repeatedly in the other window of the same cache set over the period. Whereas, the DWWR and DWAWR redirect the writes in the same cache set in a pseudo-random fashion by taking into account of the window/way write counts from the previous interval.

• **EqualChance:** The improvement by the proposed approaches (row 7-9 and 19-21) against EqualChance is due to the transfer/swap policy adopted by the existing method. In particular, for EqualChance, the transfer/swap operation takes place only when the clean/invalid data entry is present in the cache set (which is very limited). On the other hand, in our techniques, the write redirection takes place to the Least Recently Used entry (always available) in the same cache set.

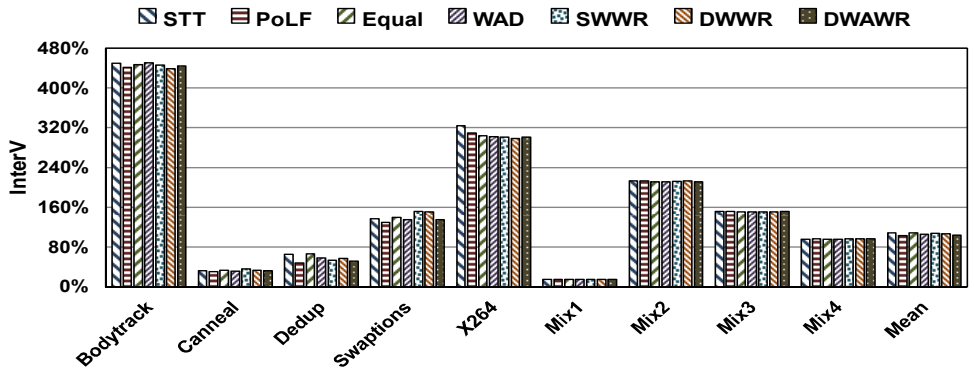


FIGURE 4.9: Inter-Set write variation for quad core (lower is better)

• **WAD:** With respect to WAD, the improvement (row 10-12 and 25-27) is mainly due to the migration of the block in a pseudo probabilistic fashion by the existing technique. In particular, the WAD probabilistically assumes that upon the write saturation of a counter, the next block to be written in the cache set is hot.

In our analysis, we also observed that some of the rows (row 14 and 29) of the table indicate the limited reduction in intra-set write variation in SPEC compared to PARSEC. The reason behind this is the limited amount of data sharing between the multiple cores in the SPEC benchmarks [6] [119] compared to multi-threaded PARSEC applications. Also, with the limited number of cores, the percentage of shared data is less (row 15 and 30). All these factors limit the write-backs from the other cores in the write-restricted window/ways.

4.4.1.2 Coefficient of Inter-Set Write Variation

Figure 4.9 and Table 4.7 shows the coefficient of inter-set write variation for quad-core and the percentage reduction in the coefficient values against the existing techniques. From the results, we observe that the inter-set write variation does not change as our proposed techniques do not redirect or move the data from one cache set to another cache set.

Core	Reference Policy	SWWR	DWWR	DWAWR	Row
Dual	STT	1.12%	3.7%	5.3%	1
	PoLF	0.3%	2.85%	4.45%	2
	Equal	-0.3%	2.3%	3.9%	3
	WAD	-3.1%	-0.5%	1.1%	4
Quad	STT	0.75%	1.3%	4.2%	5
	PoLF	-4.8%	-4.3%	-1.4%	6
	Equal	0.5%	1.1%	4%	7
	WAD	-1.9%	-1.4%	1.5%	8

TABLE 4.7: Percentage reduction in coefficient of Inter-Set write variation (higher is better)

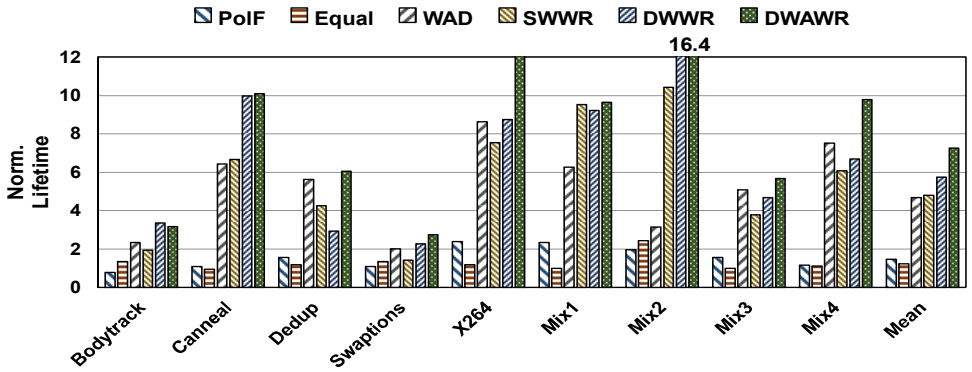


FIGURE 4.10: Normalized lifetime with respect to baseline STT-RAM for quad core (higher is better)

4.4.1.3 Relative Lifetime

Figure 4.10 shows the improvement in the lifetime by the proposed approaches against baseline STT-RAM for quad-core. Table 4.8 lists these improvement values against the existing techniques and the baseline. The reason behind the large lifetime improvement (row 1 to 12 and 16 to 27) is due to the significant reduction in coefficient of intra-set write variation (as presented in table 4.6). However, we observe that with the higher core count, the improvement is large compared to the previous work: SWWR by the proposed approaches: DWWR and DWAWR (row 13 to 15 and 28 to 30). This is because, with the higher core count, data sharing between the multiple cores increases. This gives the proposed dynamic approaches for more opportunity to control the large number of write-backs.

Core	Reference Policy	Suites	SWWR	DWWR	DWAWR	Row
Dual	STT	PARSEC	5.1	6.5	7.44	1
		SPEC	5.11	5.29	6.38	2
		Total	5.1	5.72	6.77	3
	PoLF	PARSEC	2.31	2.94	3.38	4
		SPEC	2.44	2.52	3.04	5
		Total	2.4	2.68	3.17	6
	Equal	PARSEC	2.62	3.33	3.83	7
		SPEC	4.50	4.66	5.62	8
		Total	3.65	4.1	4.85	9
	WAD	PARSEC	0.87	1.11	1.28	10
		SPEC	1.34	1.39	1.68	11
		Total	1.14	1.28	1.51	12
	SWWR	PARSEC	-	1.27	1.46	13
		SPEC	-	1.03	1.24	14
		Total	-	1.12	1.32	15
Quad	STT	PARSEC	3.6	4.56	5.78	16
		SPEC	6.92	7.7	9.7	17
		Total	4.8	5.75	7.27	18
	PoLF	PARSEC	2.77	3.52	4.47	19
		SPEC	4.04	4.5	5.65	20
		Total	3.3	3.92	4.96	21
	Equal	PARSEC	2.97	3.78	4.80	22
		SPEC	5.4	6	7.53	23
		Total	3.87	4.63	5.86	24
	WAD	PARSEC	0.83	1.05	1.34	25
		SPEC	1.31	1.46	1.84	26
		Total	1.02	1.22	1.54	27
	SWWR	PARSEC	-	1.27	1.61	28
		SPEC	-	1.11	1.4	29
		Total	-	1.2	1.51	30

TABLE 4.8: Relative lifetime improvement (in times) (higher is better)

4.4.1.4 Performance

Figure 4.11 shows the normalized CPI (with respect to STT) for quad-core. Our proposals maintain the same performance with PoLF, EqualChance, and WAD. The reason behind having the same performance despite lesser invalidations (reported in the next subsection at point 2) can be accounted for the extra cycles taken by the swap operations during the write redirection process. Whereas, there is no performance loss despite the lesser available capacity at each epoch over the baseline STT is due to the invalidation of the LRU blocks from the other windows in the same cache set. However, there is a trade-off between the number of ways or window size and performance. Lesser the number of the ways available for the allocation, larger the impact on performance. Vice-versa in the opposite case. The more elaborated discussion is given at section 4.5.2, where the size of the write restricted window/ways is altered, and its impact on EDP is seen on the table 4.15.

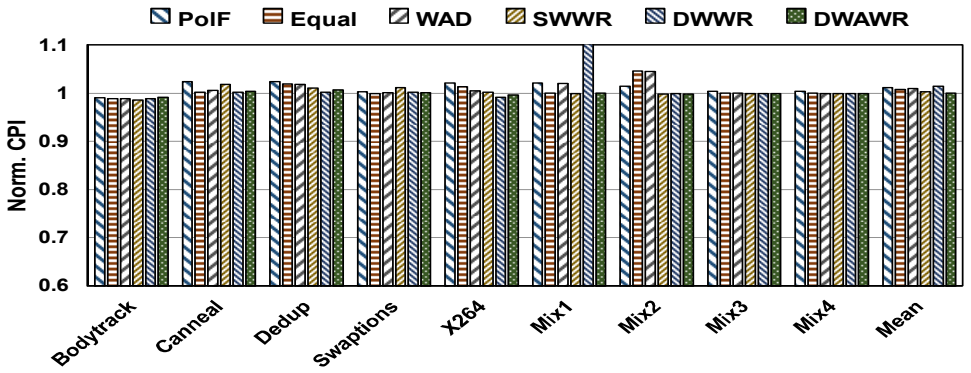


FIGURE 4.11: Normalized speedup with respect to baseline STT-RAM for quad core (lower is better)

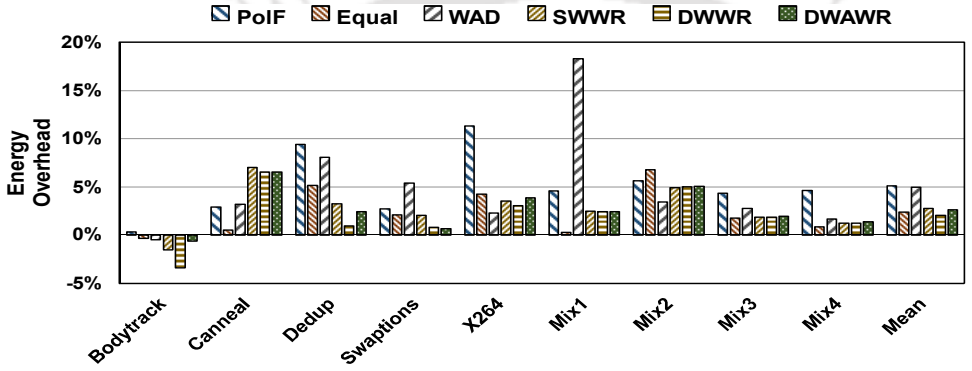


FIGURE 4.12: Energy overhead with respect to baseline STT-RAM for quad core (lower is better)

4.4.1.5 Overheads

1. ***Energy Overhead:*** As the proposed techniques redirect the writes by invalidating the data block, they need slightly more energy (row 1 and 5) compared to the baseline. Figure 4.12 presents the energy overhead by the proposed techniques against the existing methods and the baseline. Table 4.9 lists these values against the baseline and the existing methods. Note that the negative values in the table signifies the energy savings. The reasons behind the energy improvements against the existing techniques are presented below:

- **PoLF:** The energy improvement by the proposed approaches (row 2 and 6) with respect to the PoLF is due to two reasons: First, the invalidation of the MRU block by the PoLF that increases the allocations in the cache. Second,

Core	Reference Policy	SWWR	DWWR	DWAWR	Row
Dual	STT	2.36%	2.05%	1.82%	1
	PoLF	-2.46%	-2.61%	-2.85%	2
	Equal	-1.61%	-1.91%	-2.13%	3
	WAD	-0.61%	-0.91%	-1.13%	4
Quad	STT	2.78%	2.1%	2.65%	5
	PoLF	-2.16%	-2.83%	-2.28%	6
	Equal	-1.02%	-1.6%	-1.03%	7
	WAD	-1.88%	-2.55%	-1.99%	8

TABLE 4.9: Energy overhead (in percentage) (lower is better)

the number of invalidation by the proposed approaches is less as compared to PoLF.

- **EqualChance:** In case of EqualChance, the energy improvement (row 3 and 7) is due to transfer/swap operation performed by the existing method within the cache set. These kinds of activities incur extra writes in the cache, which in turn increases the energy consumption. However, in quad-core, the energy improvement is limited because of the less availability of clean entries (due to increase in the number and the residency of a dirty block) which in turn reduces the write operations.
- **WAD:** The energy improvement (row 1 and 8) by the proposed approaches over WAD is due to migration of the block in either clean LRU or the LRU position by existing proposal which incurs extra write operations in the cache that results into extra energy.

Note that, in the energy overhead calculations presented above, we have considered the energy consumed during the transfer of the tag from the original location to the redirected location.

2. ***Invalidation/Flushes:*** Figure 4.13 present the normalized invalidations by the WAD and proposed approaches against PoLF. Table 4.10 lists the percentage reduction in invalidation by the proposed techniques: DWWR and DWAWR against the existing techniques: PoLF, WAD, and SWWR. Note that the negative values in the table implies the increase in invalidation (row 2, 3, 5, and 6). The improvement in the invalidation with respect to PoLF is due to selective invalidation done by the proposed approaches. However, compared to SWWR,

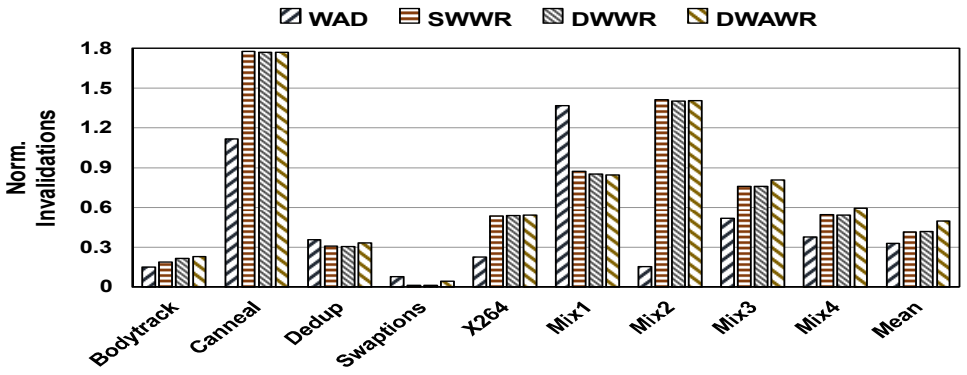


FIGURE 4.13: Normalized invalidation against PoLF for quad core (lower is better)

Core	Reference Policy	SWWR	DWWR	DWAWR	Row
Dual	PoLF	57.4%	57.8%	52.8%	1
	WAD	0.8%	1.74%	-9.97%	2
	SWWR	-	-0.96%	-11.93%	3
Quad	PoLF	57.7%	58.5%	50%	4
	WAD	-28.4%	-25.9%	-51.8%	5
	SWWR	-	-2%	-20.6%	6

TABLE 4.10: Percentage reduction in invalidations (higher is better)

the invalidation is increased as the increase of write-back operations in write restricted window/ways from L1 cache to L2 cache in the proposed approaches. On the other hand, with respect to WAD, the invalidation by the proposed approaches is large (row 2 and 5) because, in the existing method, the block is migrated to either invalid location or the clean LRU block position only when the RSC counter saturates.

4.4.2 Three Level Cache Analysis: L2-SRAM, L3-ReRAM

To evaluate our proposed techniques: SWWR, DWWR, and DWAWR in the three-level cache, we used 8MB 16-way set-associative ReRAM based L3 cache for dual-core and 16 MB 16-way set-associative ReRAM based L3 cache for quad-core. Same as in two-level cache analysis, we set the parameters' values to 2 million (1 million) cycles for I and m/n to 4 in case of dual (quad) core system (Note that the comparative analysis are only limited for two-level cache as given in the section 4.5). Figures 4.14 and 4.15 plot the intra-set write variation and lifetime for

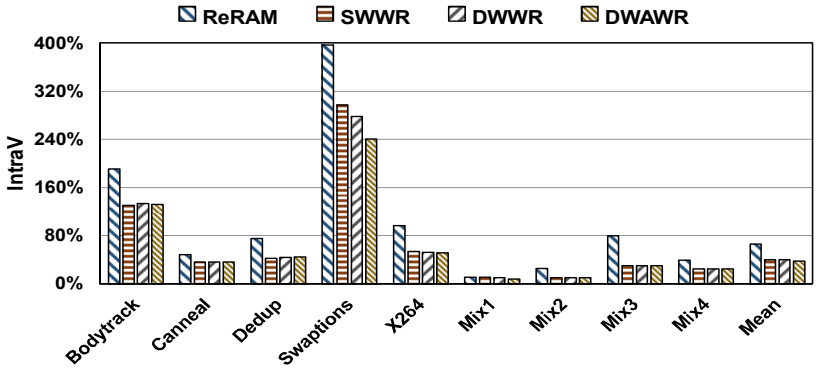


FIGURE 4.14: Intra-Set write variation for quad core ReRAM L3 cache (lower is better)

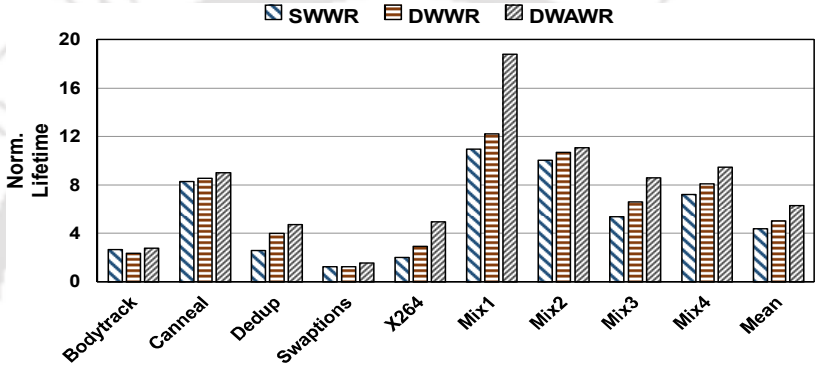


FIGURE 4.15: Normalized lifetime with respect to baseline ReRAM for quad core ReRAM L3 cache (higher is better).

quad-core system. Table 4.11 shows the brief results for the following metrics: IntraV, InterV and normalized lifetime (with respect to baseline ReRAM (presented in times)) for dual and quad-core ReRAM based L3 cache system. The limited improvement (row 1 and 4) between the proposed techniques in the intra-set write variation is due to the less write access in the L3 cache compared to an L2 cache. However, the simulation results validate that the inferences drawn out from the two-level STT-RAM cache system are still applicable to three-level ReRAM cache system.

Core	Metric	ReRAM	SWWR	DWWR	DWAWR	Row
Dual	IntraV	53.9%	27.7%	26.8%	26.5%	1
	InterV	45.2%	44.1%	42.3%	42%	2
	Lifetime	1	4.96	6.76	7.39	3
Quad	IntraV	66.7%	40.8%	40.3%	38.4%	4
	InterV	59.8%	59.8%	56%	57.9%	5
	Lifetime	1	4.41	5.04	6.31	6

TABLE 4.11: Brief results and analysis for three level ReRAM last level cache system

Core	Metric	SWWR	DWWR	DWAWR
Dual	IntraV	22.73%	22.85%	25.12%
	InterV	-6.11%	-3.54%	-1.94%
	Lifetime	1.96	2.20	2.60
	EDP	3.32%	3.6%	3.95%
Quad	IntraV	36.71%	39.65%	43.2%
	InterV	-9.43%	-8.92%	-6%
	Lifetime	2.60	3.12	3.94
	EDP	6.16%	5.72%	6.58%

TABLE 4.12: Results comparison analysis between i2wap and write restriction

4.4.3 I2WAP versus Write Restriction

Table 4.12 present the comparative analysis of InterV (percentage reduction), IntraV (percentage reduction), normalized lifetime (in times) and normalized EDP against i2wap for dual and quad-core STT-RAM based L2 cache system. Note that negative values in the table imply the increase in inter-set write variation. The respective improvements in the coefficient of intra-set write variation represent the effectiveness of the proposed approaches: SWWR, DWWR, and DWAWR. However, by integrating some of the inter-set wear-leveling techniques with the proposed methods, the lifetime and the inter-set write variation can further be improved with respect to i2wap. The gain in EDP by the proposed approaches compared to i2wap is mainly due to large random invalidations performed by the existing approach, which in turn affect the system performance and increase the energy consumption.

4.4.4 FLASH based Adaptive Wear Leveling Technique (FAWLT) versus Write Restriction

Along with the NVM based wear leveling technique, previous literature also reported wear leveling for the flash-based memories. Hence, it is worthwhile to

Core	Metric	SWWR	DWWR	DWAWR
Dual	IntraV	50.97%	51.1%	53.35%
	InterV	-2.04%	0.52%	2.12%
	Lifetime	4.08	4.57	5.41
	EDP	-2.72%	-2.42%	-2%
Quad	IntraV	67.53%	70.5%	74%
	InterV	-4.8%	-4.3%	-1.34%
	Lifetime	3.86	4.63	5.85
	EDP	-2.90%	-3.37%	-2.44%

TABLE 4.13: Results comparison analysis between FAWLT and write restriction

compare our architectures with one of the existing flash-based proposals: Adaptive Wear Leveling in the Flash-based Memory (or FAWLT) [120]. This section illustrates such a comparison analysis between flash-based wear leveling and the proposed techniques. Table 4.13 reports the comparison analysis between the write restriction and FAWLT. Note that the negative values in the table imply the increase in the EDP and inter-set write variation. The result obtained from the table shows the efficacy of the proposed techniques: SWWR, DWWR, and DWAWR. Thus, from the result, it can be concluded the wear leveling approach used for flash memory is not suitable for the caches.

4.5 Comparative Analysis for Parameters

In addition to the results presented in the previous section, we conducted experiments with the different STT-RAM based L2 cache configurations and, with the different values of the parameters used in the approaches. Here, in this section, we show the effect of altering the different parameters on various metrics as comparison to the reference parameters. This kind of analysis is very helpful for choosing the optimal values in the proposed algorithms for different configurations of the cache.

4.5.1 Change in Interval (I)

Table 4.14 reports the different metrics for distinct interval values in the proposed schemes. Change in the interval affects the frequency of the selection process of

Core	Policy	Param.	Norm. Lft.	IntraV Base	IntraV WrRes	InterV Red. (%)	Norm. EDP	Invalid-ation (k)	Row
Dual	SWWR	Ref.(I=2M)	5.1	90.94%	34.2%	1.12%	1.02	433k	1
		I=1M	5.36	90.94%	33.5%	3.7%	1.02	483k	2
		I=5M	4.38	90.94%	35.6%	0.06%	1.01	364k	3
	DWWR	Ref.(I=2M)	5.72	90.94%	34.1%	3.7%	1.02	437k	4
		I=1M	5.68	90.94%	33.9%	2.4%	1.03	483k	5
		I=5M	4.96	90.94%	34.8%	0.2%	1.02	382k	6
	DWAWR	Ref.(I=2M)	6.77	90.94%	31.8%	5.3%	1.02	485k	7
		I=1M	6.18	90.94%	31.8%	4%	1.03	555k	8
		I=5M	5.57	90.94%	33.1%	-1.2%	1.02	420k	9
Quad	SWWR	Ref.(I=1M)	4.8	139.7%	59.7%	0.75%	1.02	490k	10
		I=0.5M	5.3	139.7%	59.8%	2.74%	1.03	558k	11
		I=2M	4.4	139.7%	61.6%	2%	1.01	430k	12
	DWWR	Ref.(I=1M)	5.75	139.7%	56.7%	1.26%	1.03	500k	13
		I=0.5M	6.1	139.7%	57.7%	1.95%	1.03	568k	14
		I=2M	4.87	139.7%	59.7%	4.3%	1.01	438k	15
	DWAWR	Ref.(I=1M)	7.27	139.7%	53.2%	4.2%	1.03	591k	16
		I=0.5M	5.93	139.7%	54.1%	4.4%	1.03	674k	17
		I=2M	6.45	139.7%	56.8%	2.2%	1.01	525k	18

TABLE 4.14: Comparative analysis for different interval values (I) LFT.= lifetime, BASE = baseline STT-RAM, WrRes = Write Restricted

write restricted window/ways in the cache. With large interval value (row 3, 6, 9, 12, 15, and 18), the frequency of window/ways selection process is low compared to the reference case. This, in turn, lessens the premature invalidation from the L2 cache because the block evicted from other windows/ways are mostly LRU blocks compared to the reference case. However, the long interval value leads to the bigger coefficient of intra-set write variation with lesser improvement in the lifetime. On the other hand, the smaller interval value (row 2, 5, 8, 11, 14 and 17) increases the write restricted window/ways selection frequency which in turn increases the invalidations in the cache. Also, the smaller interval value does not capture all the write-backs from the L1 cache to the write restricted window/ways. Thus, the coefficient of intra-set variation and relative lifetime is not improved as much as compared to the reference case. In addition, with the smaller interval, due to premature evictions of the block, the system performance is severely affected with increased EDP (shown in column 8 of row 2, 5, 8, 11, 14 and 17 of Table 4.14).

4.5.2 Change in Write Restricted Window/Ways (m/n)

Table 4.15 lists the result metrics for different write restricted window/ways sizes (m/n). By altering the size of write restricted windows/ways, the write redirection

Core	Policy	Param.	Norm. Lft.	IntraV Base	IntraV WrRes	InterV Red. %	Norm. EDP	Invalid-ation (k)	Row
Dual	SWWR	Ref.(m=4)	5.1	90.94%	34.2%	1.12%	1.02	433k	1
		m=2	5.2	90.94%	33.7%	1.55%	1.03	410k	2
		m=8	4.7	90.94%	37.8%	-0.22%	1.00	309k	3
	DWWR	Ref.(m=4)	5.72	90.94%	34.1%	3.7%	1.02	437k	4
		m=2	5.96	90.94%	33.3%	1.68%	1.03	585k	5
		m=8	4.73	90.94%	37.4%	0.85%	1.00	313k	6
	DWAWR	Ref.(n=4)	6.77	90.94%	31.8%	5.3%	1.02	485k	7
		n=2	4.9	90.94%	36.6%	0.72%	1.01	325k	8
		n=8	6.83	90.94%	31.4%	3%	1.04	590k	9
Quad	SWWR	Ref.(m=4)	4.8	139.7%	59.7%	0.75%	1.02	490k	10
		m=2	5.92	139.7%	58.6%	0.74%	1.04	600k	11
		m=8	4.14	139.7%	64.8%	-1.97%	1.02	368k	12
	DWWR	Ref.(m=4)	5.75	139.7%	56.7%	1.26%	1.03	500k	13
		m=2	6.14	139.7%	56.5%	5.1%	1.04	602k	14
		m=8	4.24	139.7%	65.6%	-1.15%	1.03	385k	15
	DWAWR	Ref.(n=4)	7.27	139.7%	53.2%	4.2%	1.03	591k	16
		n=2	5.1	139.7%	63.1%	-1.71%	1.02	143k	17
		n=8	7.77	139.7%	51.7%	3.1%	1.04	399k	18

TABLE 4.15: Comparative analysis for different write restricted window/ways size (m/n) LFT.= lifetime, BASE = baseline STT-RAM, WrRes = Write Restricted

process of the proposed approaches is influenced. On increasing the size (row 2, 5, 9, 11, 14, and 18), the number of write redirection increase (as more ways are selected) which in turn improves the lifetime and reduces the coefficient of intra-set write variation more than the reference case. However, at the same time, it increases the invalidations (due to less availability of ways in the other window/ways) and also increases the EDP. Besides, on reducing the size (row 3, 6, 8, 12, 15 and 17), fewer ways are selected for the write redirection process and this, in turn, reduces the lifetime improvement and increases the coefficient of intra-set write variation.

4.5.3 Change in Capacity

Table 4.16 shows the behavior of the proposed schemes with different cache capacities. Larger cache (row 3, 6, 9, 12, 15 and 18) suffers from the significant intra-set write variation as compared to smaller cache (row 2, 5, 8, 11, 14 and 17). This is because, in case of larger cache, the cache block faces less capacity miss as compared to smaller cache. This, in turn, increases the residency of the block and increases the intra-set write variation more than the reference case. However, in these cases, for small and large size cache, the proposed schemes effectively reduce

Core	Policy	Param.	Norm. Lft.	IntraV Base	IntraV WrRes	InterV Red. %	Norm. EDP	Invalid-ation (k)	Row
Dual	SWWR	Ref.(8MB)	5.1	90.94%	34.2%	1.12%	1.02	433k	1
		4MB	5.2	76.2%	25.1%	5.92%	1.02	432k	2
		16MB	3.55	111.6%	51.5%	0.66%	1.01	334k	3
	DWWR	Ref.(8MB)	5.72	90.94%	34.1%	3.7%	1.02	437k	4
		4MB	5.65	76.2%	24.7%	6.37%	1.03	439k	5
		16MB	4.79	111.6%	49.2%	0.05%	1.02	339k	6
	DWAWR	Ref.(8MB)	6.77	90.94%	31.8%	5.3%	1.02	485k	7
		4MB	6.25	76.2%	23.6%	6.78%	1.03	447k	8
		16MB	4.81	111.6%	47.4%	0.53%	1.02	385k	9
Quad	SWWR	Ref.(16MB)	4.8	139.7%	59.7%	0.75%	1.02	490k	10
		8MB	5.1	112.5%	43.1%	4.47%	1.01	612k	11
		32MB	5.3	163.1%	78.1%	14.5%	1.02	414k	12
	DWWR	Ref.(16MB)	5.75	139.7%	56.7%	1.26%	1.03	500k	13
		8MB	5.5	112.5%	41.5%	2.4%	1.02	617k	14
		32MB	6	163.1%	76.9%	17.2%	1.02	423k	15
	DWAWR	Ref.(16MB)	7.27	139.7%	53.2%	4.2%	1.03	591k	16
		8MB	5.8	112.5%	38.3%	5%	1.02	703k	17
		32MB	6.23	163.1%	72.8%	15.5%	1.02	517k	18

TABLE 4.16: Comparative analysis for different LLC capacity

the coefficient of intra-set write variation and improve the relative lifetime very efficiently.

4.5.4 Change in Associativity

Table 4.17 shows the different behaviors by the proposed schemes under different associativity of L2 cache. Cache with higher associativity (row 3, 6, 9, 12, 15 and 18) suffers from large intra-set write variation as compared to cache with lower associativity (row 2, 5, 8, 11, 14 and 17). This is because cache associativity impacts the conflict misses, which in turn affects the residency of the block. In particular, the cache with large associativity faces fewer conflict misses as compared to cache with lower associativity. This increases the residency of the block in case of cache with higher associativity. In both the cases (cache with lower and higher associativity), our proposed schemes reduce the coefficient of intra-set write variation and improve the lifetime.

4.5.5 Storage Overhead

Our techniques incur the limited amount of storage overhead over the baseline STT-RAM. The amount of overheads by SWWR is only 42 bits. However, the

Core	Policy	Param.	Norm. Lft.	IntraV Base	IntraV WrRes	InterV Red. %	Norm. EDP	Invalid-ation (k)	Row
Dual	SWWR	Ref.(A=16)	5.1	90.94%	34.2%	1.12%	1.02	433k	1
		A=8 way	2.78	69.94%	32%	1.94%	1.02	407k	2
		A=32 way	6.73	109.6%	37.5%	3.25%	1.04	462k	3
	DWWR	Ref.(A=16)	5.72	90.94%	34.1%	3.7%	1.02	437k	4
		A=8 way	2.91	69.94%	30.3%	1.92%	1.02	412k	5
		A=32 way	8.53	109.6%	36.1%	2.1%	1.04	464k	6
	DWAWR	Ref.(A=16)	6.77	90.94%	31.8%	5.3%	1.02	485k	7
		A=8 way	3.40	69.94%	29.2%	2.3%	1.02	425k	8
		A=32 way	8.78	109.6%	34.5%	2.3%	1.04	508k	9
Quad	SWWR	Ref.(A=16)	4.8	139.7%	59.7%	0.75%	1.02	490k	10
		A=8 way	3.16	109.8%	50.6%	2.8%	1.02	491k	11
		A= 32 way	6.40	168.8%	66.4%	2.3%	1.03	522k	12
	DWWR	Ref.(A=16)	5.75	139.7%	56.7%	1.26%	1.03	500k	13
		A=8 way	3.62	109.8%	48.8%	6.3%	1.02	494k	14
		A=32 way	7.53	168.8%	64.7%	3.8%	1.04	568k	15
	DWAWR	Ref.(A=16)	7.27	139.7%	53.2%	4.2%	1.03	591k	16
		A=8 way	3.84	109.8%	45.5%	7.55%	1.02	545k	17
		A=32 way	7.92	168.8%	62.4%	-0.34%	1.04	606k	18

TABLE 4.17: Comparative analysis for different LLC associativity (A)

		DWWR		DWAWR	
Param.	Core	C_i (bits)	Overhead (bits)	Z_k (bits)	Overhead (bits)
Reference	Dual	13	94	11	218
	Quad	13	94	11	218
I=5M/2M	Dual	15	102	12	234
	Quad	14	98	12	234
I=2M/0.5M	Dual	12	90	10	202
	Quad	11	86	10	202
m=2/n=8	Dual	14	98	11	218
	Quad	16	106	11	218
m=8/n=2	Dual	12	90	11	218
	Quad	11	86	11	218
4MB/8MB	Dual	12	90	10	202
	Quad	11	86	9	186
16MB/32MB	Dual	16	106	14	266
	Quad	15	102	12	234
8way	Dual	10	82	8	170
	Quad	9	78	7	154
32way	Dual	18	114	16	298
	Quad	17	110	15	282

TABLE 4.18: Counter sizes and storage overhead (in bits) of DWWR and DWAWR

window counter (C_i) and way counter (Z_k) sizes in case of DWWR and DWAWR are depend upon the cache configuration and the parameters of the algorithms. Table 4.18 reports the counter size and storage overhead for the different configurations and parameters of the algorithms with respect to the chosen configuration: 8MB/16MB 16-way set-associative L2 cache, $I = 2M/1M$ and $m/n = 4$ on the dual/quad-core system. The depicted overheads are mainly due to the window and way counter size and the 42-bit tag buffer.

	PARSEC v2.1					SPEC CPU 2006			
Workload	**Body**	**Cann**	**Dedup**	**Swap**	**X264**	**Mix1**	**Mix2**	**Mix3**	**Mix4**
STT-RAM									
Ideal	41.8K	3.8K	4.8K	14.3K	8.7K	3.21K	7.44K	14.9K	25.5K
Baseline	38	19.2	30.9	41.3	8.65	47.3	6.33	25.8	45.7
SWWR	72.9	128.5	134.6	60.9	65.5	451.1	66	98.1	278.5
DWWR	125.5	191.6	91.4	94.1	75.1	436.5	76.5	121	306.9
DWAWR	119.5	194.1	189.4	114.2	104.6	456.4	103.9	146.6	447.8
ReRAM									
Ideal	2.12K	99.5	795.4	6.37K	462.2	185.4	175.2	565.5	666.2
Baseline	0.61	0.51	0.53	1.17	1	1.16	2.22	3.03	3.02
SWWR	1.63	4.26	1.48	1.48	2.05	13.3	22.51	16.34	21.86
DWWR	1.44	4.43	2.10	1.45	2.96	14.87	22.1	20.13	24.6
DWAWR	1.71	4.63	2.58	1.82	5.10	22.85	24.8	26.2	28.77

TABLE 4.19: Lifetime comparison analysis (in years) by the proposed schemes: SWWR, DWWR and DWAWR.

	SWWR/DWWR		DWAWR	
Cache Configuration	**m**	**I**	**n**	**I**
Small Assoc., Small Size	=	= / ↑	↓	= / ↑
Small Assoc., Large Size	=	= / ↑	↓	= / ↑
Large Assoc., Small Size	=	↓	↑	↓
Large Assoc., Large Size	↓	↓	↑	↓

TABLE 4.20: Recommended values of m and I with respect to reference values $m = n = 4$ and $I = 1M$ (dual) $= 2M$ (quad) cycles.

4.5.6 Lifetime Comparison Analysis

Table 4.19 presents the lifetime comparison analysis (in years) over the ideal STT-RAM/ReRAM (where the writes are uniformly distributed), baseline STT-RAM/ReRAM, (with no wear leveling policy as reported in table 2.3) and our proposed schemes: SWWR, DWWR, and DWAWR. Note that the values presented in the table are calculated from the reference value presented in the section 4.4 (For this analysis, the write endurance values are taken as 4 x 10^{12} and 10^{11} writes for STT-RAM and ReRAM respectively as given in [121, 20]). The conclusion that can be derived from the table is that our technique: DWAWR works better than the SWWR and DWWR for both types of NVMs.

4.5.7 Recommended Values

Based on the above-detailed analyses, we recommend the values to be used for m, n and I for the different configurations of caches. Table 4.20 lists these recommended values for the proposed schemes: SWWR, DWWR, and DWAWR. Note that these

recommended values are with respect to $m = n = 4$ and $I = 2M$ (for dual-core), $1M$ (for quad-core) cycles. The rationale behind the recommendations is presented below:

• **Interval (I):** The value of I depends upon the associativity. For the cache with smaller associativity, to capture the maximum write-backs, the value of I can be kept same or increased (according to requirement). On the other hand, in the case of cache with larger associativity, to control the block residency and to reduce the intra-set variation, the value of I needs to be reduced.

• **Write Restricted Window/Ways (m/n):** The value of m, n is decided based upon the following configuration of cache:

– **Small Associativity:** Smaller associative cache needs the small number of write restricted ways. Hence, the value of n should reduce in case of DWAWR. In the case of SWWR and DWWR, the value of m can be the same; however, the window size (regarding ways) will reduce as associativity is small.

– **Large Associativity, Small Size:** In this case, to control the block residency, the value of n (Write restricted ways) needs to be increased. On the other hand, in the case of SWWR and DWWR, the partition size is increased accordingly with the number of partitions (m). Hence, the value of m is kept the same.

– **Large Associativity, Large Size:** To control the large intra-set write variation, in this case, the partition (SWWR/DWWR)/way (DWAWR) (n) (DWAWR) size need to be increased, or the number of partitions (m) has to be reduced.

4.6 Summary

Due to the differing working set sizes and run-time access patterns of applications, the non-volatile caches suffer from write variations. These write variations not only reduce the lifetime but also shrink the capacity of cache over the period. Write variations inside the cache are governed by the access pattern as well as the

replacement policies. In this chapter, we presented three techniques to mitigate the intra-set write variation. Our first two techniques partition the cache into multiple windows and uses a different window during the execution as write-restricted or read-only window. In our first technique, the window is selected in a round robin fashion. On the other hand, in our second technique, the window is selected by considering its write intensity over a period of execution. In the third technique, instead of partitioning the cache into windows, a set of cache ways are selected based on their write-intensity. The selected ways are treated as write restricted or read-only for certain predefined interval.

The efficacy of the proposed schemes is examined with the help of three existing techniques: PoLF, WAD and EqualChance and, the baseline STT-RAM/ReRAM with no wear leveling policy support. Experimental results show the significant reduction in the coefficient of intra-set write variation along with the improvement in the relative lifetime for dual and quad-core systems. Thus, if we reduce the non-uniform write distribution inside the cache set, we can effectively utilize the emerging non-volatile memories in hardware systems.

Chapter 5

Intra-Set Wear Leveling using Write Restricted Horizontal Partitions

This chapter proposes another alternative intra-set wear leveling method for lifetime longevity enhancement of non-volatile caches. As same as previous chapter, the proposed method works on the basic concept of write restriction. However, the only difference is that proposed technique is applicable at the different granularity of the cache bank. In particular, the proposed method divides the cache logically into multiple equal-sized modules. During execution, the writes are uniformly distributed across different ways of the different modules within the cache set. The proposed technique are evaluated on two different existing methods with quad-core system.

5.1 Introduction

In this chapter, to control intra-set write variation with less performance and energy overheads, we make different sub-ways of the different sets as write-restricted, i.e., read-only over an interval. The proposed method logically divides the cache

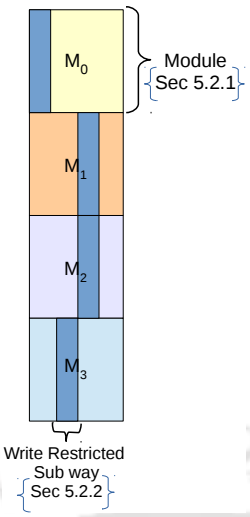


FIGURE 5.1: General overview of contribution in chapter 5

into multiple equal-sized parts called modules in such a way that each module contains an equal number of sets. During the execution, for a certain predefined interval, different m ways are chosen from each module and made write-restricted. At the end of the interval, the next m ways are chosen and treated as a write-restricted for the next interval, and the process continues until the end of execution. Note that each module of the cache can have different m ways selected. In this chapter, we applied our proposed technique on STT-RAM-based LLC (i.e., L2 cache). Although, the technique can be easily extended to ReRAM and PCRAM based NVM cache. The main contributions of this chapter are as follows:

- The chapter presents a technique to mitigate the intra-set write variation and improve the lifetime of non-volatile based cache. It divides the cache into multiple modules and controls write variation within each module.

- Experimental evaluation is performed over the full system simulator GEM5 [118] and results are compared with two existing techniques: Polf [29], WriteSmoothing [33] and the baseline STT-RAM with no wear-leveling associated.

- In-depth analysis with different configurations of L2 cache as LLC and, with the different parameters of the method are also presented.

Figure 5.1 presents the general overview of the proposed contribution in this chapter.

The chapter is organized as follows: section 5.2 presents the proposed wear leveling method. The experimental methodology is illustrated in section 5.3. Results and analysis are discussed in section 5.4. Section 5.5 reports the parameter comparison analysis. Finally, we summarize this chapter in section 5.6.

5.2 Proposed Wear Leveling Technique: MWWR

This section discusses our proposed wear leveling technique called Module Wise Write Restriction (MWWR).

5.2.1 Architecture

The key idea of the proposed methodology is to logically divide the cache into multiple equal-sized modules or parts (in such a way that each module contains an equal number of sets) and use m different ways in each module at the regular intervals of the application execution. We call these ways inside the module as sub-ways. During an interval, the selected sub-ways in a module are treated as a write-restricted (or read-only) sub-ways. In particular, all the write requests from the L1 cache to the selected sub-ways of the module are redirected to other ways (other than the selected sub-ways) of the module within the same cache set. In each interval, the selection of sub-ways in the module is based upon a counter associated with each sub-way of the module. The use of the counter is to track the number of L1 write accesses that the particular sub-way entertained in the past interval(s). In each interval and for each module, the m sub-ways with maximum counter values are chosen and treated as a write-restricted sub-ways. At the end of the interval, the next m sub-ways of each module are chosen, and the process continues until the end of the execution. Note that, in order to remove the possibility of choosing the same sub-way(s) within the module in the successive

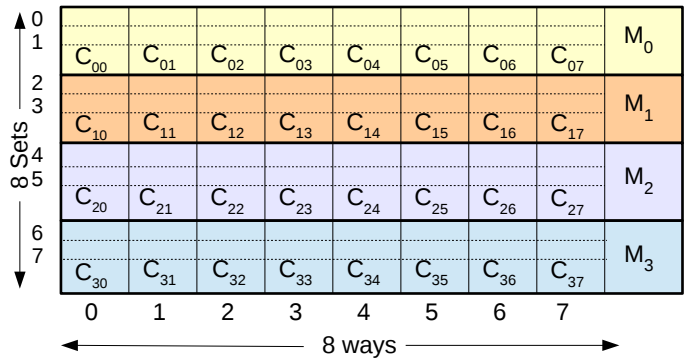


FIGURE 5.2: Example of the proposed MWWR L2 cache architecture

intervals, the sub-way counter of each selected sub-way is reset to zero. Note that other lightly written sub-ways counters are remained to be intact. The reason behind to initialize the counter with zero ensure that all the other lightly written sub-ways counters get the chance for the write restriction over the intervals by accumulating the writes in the past intervals. Also, note that these m sub-ways (chosen for write restriction) may not be contiguous.

Consider a set-associative cache with S number of cache sets and associativity A. Let the total number of modules in the cache is M then each module in the cache has following sets of attributes:

- Total number of sets in the module (S_{mod}): S/M.
- All the sub-ways-j of sets in a module share the same counter. Hence, the total number of counters in a module: A.
- Total number of counters in a cache bank: $A * M$.
- Module label for a given set-id (C_{set}) of a cache bank: $\lfloor C_{set}/S_{mod} \rfloor$.

The example of our architecture is presented in fig. 5.2. As shown in the figure, an 8-way set associative L2 cache ($A = 8$) having eight sets ($S = 8$) and four modules ($M = 4$) is considered to demonstrate the example. The four modules of the cache are labeled with M_0, M_1, M_2 and M_3. With the given values, the $S_{mod} = S/M = 2$ cache sets are present inside the module and a total of $A * M = 32$ counters are

needed to track the write accesses of each sub-ways of the modules inside a cache bank in the current interval or in the past intervals until it is reset to zero. In the figure, these sub-way counters are represented by variable C_{ij} where i represents the module-id, and j denotes the sub-way id. For example, C_{04} is counter for module-M_0 and sub-way 4 of all sets in M_0. Similarly, the counters C_{00} to C_{07} represent the counter for each sub-way in the module M_0.

5.2.2 Operation

The operation of the proposed wear leveling technique is elaborated through algorithm 4. In the algorithm, the parameter I acts as a tunable value for the predefined interval (line 1). The total number of sub-ways that are treated as a write-restricted in each module is represented by the variable m (line 2). The variable M represents the total number of modules inside a cache bank (line 3). $ModulewaysList$ is a list of lists of size $M * m$. It contains the m sub-ways id of all the M modules which are treated as write-restricted in that particular period of the regular interval I (line 4). Thus, the $ModulewaysList$ is populated in each interval. The counter associated with each sub-way of the module is represented by the variable C_{ij} (line 5).

For the initial I cycles of process execution, the cache bank is treated as a normally available bank (line 6). Once the process executes the I cycles, different sub-ways of the different modules are treated as a write-restricted for the next I cycles (line 7). The selection of sub-ways in each module is based upon the counter C_{ij} associated with the sub-ways of the modules. In particular, from each module, maximum m sub-ways counter values are chosen and placed into the $ModulewaysList$ of that particular module (line 12 and 13). By this way, the proposed technique restricts the chances of heavily written sub-ways of the modules to get further more writes in the next interval. Once the m sub-ways of the module are selected for write restriction, the respective counters associated with them are reset to zero (line 14). When the interval I ends, the write restricted sub-way list is prepared again for each module and the process continues until the end of execution.

Algorithm 4 Wear Leveling Algorithm - MWWR

1: I : Predefined interval.
2: m : Number of sub-ways in each module that are treated as read only or write restricted.
3: M: Number of modules in the cache.
4: List $< integer, List < integer >> ModulewaysList$: List of write restricted sub-ways in each module. Size of list is $M * m$.
5: C_{ij} : Sub-way counter with respect to i^{th} module and j^{th} sub-way that records the number of write accesses from L1 cache to that particular sub-way. $0 \leq i < M$, $0 \leq j < cache\_assoc$.
6: Run application for I cycles treating the whole cache as normally available cache.
7: After the I cycles treat m ways of each module as read-only or write restricted.
8: **repeat**
9: **for** every interval I **do**
10: **for** $k \leftarrow 0$ to M **do**
11: **for** $l \leftarrow 0$ to m **do**
12: Let C_{ij} be the maximum counter among all the counters in the module i of cache. $0 \leq j < cache\_assoc$
13: $ModulewaysList[k][l] = j$ ▷ **create module list of heavily written ways**
14: $C_{ij} = 0$
15: **end for**
16: **end for**
17: **for** each request R from L1 cache to the block B in L2 cache during I cycles **do**
18: **if** $R = ReadHit$ **then**
19: Perform the read operation as in the conventional cache.
20: **else if** $R = WriteHit$ **then**
21: **if** the request R is for the block B that belongs to the current $ModulewaysList$ **then**
22: The write request for the block B is redirected to the other location L in the same cache set. Note that $L \notin ModulewaysList$.
23: The corresponding sub-way counter C_{ij} of the location L is incremented.
24: **else**
25: Write operation is performed on B as in conventional cache. Increment the counter C_{ij} of the way where the block B is present.
26: **end if**
27: **else**
28: Forward the Request R to main memory to fetch the block. Keep the newly arrived block in a location L such that $L \notin ModulewaysList$. ▷ **cache miss**
29: **end if**
30: **end for**
31: **end for**
32: **until** the end of the execution

In the meanwhile, between the intervals, for each request R coming from L1 cache to the L2 cache, the tag lookup operation is performed for that particular request in the L2 cache (line 17). Based upon the result of lookup operation and the type of requests, different operations are performed in the L2 cache which is as follows (Note that the PUTX used in the following cases represents the write-back of dirty data from the L1 cache to L2 cache):

- **Read Hit:** The read operation is performed for the requested block B as same as in the conventional cache (line 18 and 19).

- **Write Hit (Write-back or PUTX) and block B not in $ModulewaysList$:** The write request R is served from the original location of block B. Once request is served, the sub-way counter of the module where block B is located is incremented (line 24 to 26).

• **Write Hit (Write-back or PUTX) and block B in $ModulewaysList$:** If the requested block B is present in the cache at location T that belongs to the $ModulewaysList$, then the write request R (from L1 cache) is redirected to the first invalid way of the same cache set other than the ways belonging to $ModulewaysList$. In case, if there is no invalid entry present in the other sub-ways of same cache set, one Least Recently Used (LRU) victim block is picked, say v from the location L. Note that the location L is the location other than the location contained in the $ModulewaysList$ for that particular module. Once the victim entry v is selected, the write-back operation is scheduled for v according to the status of its dirty bit. Subsequently, the request R is redirected to the location L and the block is written in that location. Once the request is served, the block B is invalidated from its location T. The counter for the sub-way where the write was redirected, L in this case, is incremented (line 20 to 23).

• **Cache Miss:** The request R from the L1 cache is forwarded to next level of memory. When the requested block has arrived, it is placed in a location other than those belonging to $ModulewaysList$ for that particular module (line 27 to 29).

The working example of the proposed MWWR wear leveling technique is depicted in fig. 5.3. As shown in the figure 5.3 (a) at time-stamp t1, in module-2 (M_2), the sub-way 1 and 5 are treated as write-restricted for the current interval ($ModulewaysList[2] = \{1, 5\}$). Three cases are considered in part(a) to demonstrate the method for M_2 i.e., set-4 and set-5 of a cache bank. In the first case, a read request (shown by arrow-1) to the sub-way 2 of set-4 is served normally by the cache (arrow-2). In the second case, the write request (shown by dotted arrow-3) from the L1 cache to set-4 and sub-way 7 is served (arrow-4) normally by the L2 cache. Once the write operation is completed, the respective counter C_{27} is incremented. For the last case, the write request (arrow-5) from the L1 cache to the set-5 and sub-way 5 (sub-way treated as write restricted in the current interval) is redirected (arrow-6) to the one of the other ways (3 and 6 in our case) based upon the availability and the victim entry location in the same cache set. Depending

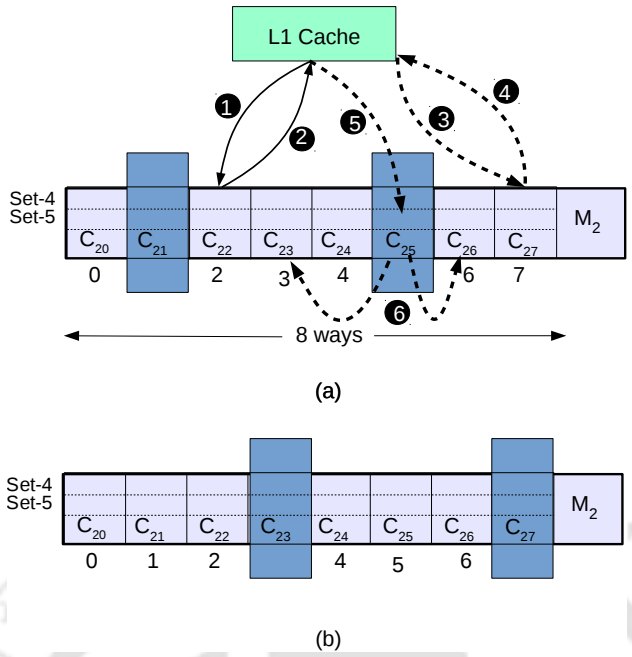


FIGURE 5.3: Working example of the proposed MWWR wear leveling technique. (a) Status of module-2 at time-stamp t1 (b) Status of module-2 at time-stamp t2

Components	Parameters
Processor	2Ghz, Quad Core, X86
L1 Cache	Private, 32 KB SRAM Split I/D caches, 4-way set associative cache, 64B block, 1-cycle latency, LRU, write-back policy
L2 Cache	Shared, 64B block, LRU, write-back policy
Main Memory	2GB, 160 cycle Latency
Protocol	MESI CMP Directory

TABLE 5.1: System parameters

upon where the write is redirected, the respective sub-way counter is incremented, and the data block in the set 5 of sub-way is invalidated because of its relocation. If the write is redirected to the sub-way 3, then the counter C_{23} is incremented. At the end of time-stamp t1 interval, for time-stamp t2 interval (shown in fig. 5.3 (b)), different sub-ways (3 and 7) are selected for write restriction based upon the values of write counters.

L2 Configuration	Leakage Power (mW)	Hit Energy (nJ)	Miss Energy (nJ)	Write Energy (nJ)	Hit Latency (ns)	Miss Latency (ns)	Write Latency (ns)
32MB, 16way	454.35	0.486	0.094	4.215	5.047	1.616	12.425
16MB, 32way	423.03	0.534	0.193	6.296	4.19	1.522	11.974
16MB, 16way	406.22	0.432	0.092	4.162	4.227	1.560	11.974
16MB, 8way	405.40	0.387	0.047	3.189	4.225	1.558	11.974
8MB, 16way	136.2	0.329	0.096	4.164	3.605	1.479	11.81

TABLE 5.2: Timing and energy parameters for STT-RAM L2 cache

5.3 Experimental Methodology

We implemented our proposed wear-leveling policy: MWWR on a full system simulator GEM-5 [118]. Table 5.1 shows the system parameters used in the simulations. Table 5.2 shows the timing and energy parameters for the L2 configurations obtained by NVSIM [21] at 32 nm technology node.

We evaluate our technique: MWWR against two existing techniques: PoLF and WriteSmoothing (termed as Wsmooth) and the baseline STT-RAM (STT) that uses LRU as a replacement policy with no support of wear leveling. In PoLF, the value of FT is set to 10. Whereas in Wsmooth, the limit of the maximum sub-ways to be turned off in each module is set to 3, and the parameter λ is set to 15%, the interval value is set to 5M cycles and the total number of modules in the cache are 128. All computation, latency, storage and selection overhead of the modules with transfer of cache block are modeled in the same way as similar to [33].

In MWWR, during the write redirection, the tag needs to be transferred from the current write-restricted sub-way to the new location within the cache set. The transfer of tags requires 128 additional 42-bit swap buffers and extra three cycles (1 cycle to transfer the tag from the sub-way to swap buffer, 1 cycle for writing the data into the swap buffer and 1 cycle for transferring the data from the swap buffer to the new location). The size of the counters (C_{ij}) associated with each module of the sub-way is set to 10 bit[1]. We assume that the computation or selection of the *ModulewaysList* is not on the critical path as it can be computed in the background just before the end of the current interval.

[1]Based on the experimental results, we selected the most stable sized C_{ij}

Benchmark suite	Benchmarks
PARSEC v2.1	Bodytrack (Body), Canneal (Cann), Dedup, Freqmine (Freq) X264
SPEC CPU2006	**Mix1:** perlbench, gcc, milc, hmmer **Mix2:** soplex, omnetpp, bzip2, libquantum **Mix3:** gobmk, tonto, sjeng, namd **Mix4:** calculix, astar, dealII, h264ref

TABLE 5.3: Benchmarks used for evaluation

To evaluate our simulated system, we use both multi-threaded: PARSEC [6] and multi-programmed: SPEC CPU 2006 [7] benchmark suites, Table 5.3. Note that the detailed description of the simulation framework is given at appendix A.

5.4 Results and Analysis

Out of the different configurations, MWWR examined on 16 MB 16-way set-associative L2 cache with I=5M cycles, M=128 and, m=4. In the later section, we analyze the effect of changing these values on the proposed approach. We have presented the results on the following set of metrics: coefficient of intra-set write variation ($IntraV$) (calculated with the help of eq. (2.2)), relative lifetime improvement (calculated with the help of eq. (2.3)), speedup, energy overhead, and invalidations.

5.4.1 Coefficient of Intra-Set Write Variation

Figure 5.4 shows the coefficient of intra-set write variation. Our proposed technique: MWWR reduces the intra-set write variation from 154.4% (STT), 114.6% (Polf), 88.8% (Wsmooth) to 61.4% (MWWR). The reduction in intra-set write variation over STT is basically due to uniform-write distribution by MWWR inside each module. However, the further reduction in coefficient by MWWR against Polf and Wsmooth is due to two reasons: (i) Polf invalidates the data randomly without concerning its write behavior. (ii) The MWWR selects the m hot sub-ways for the write restriction in each interval, and with Wsmooth, the hot-sub-way are turned off only when the module's intra-set write variation increases beyond

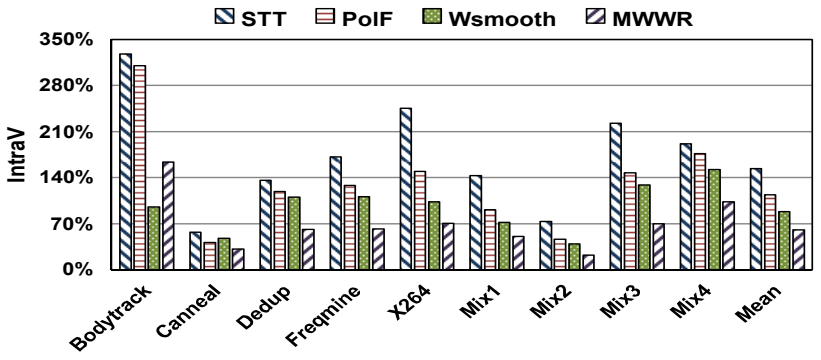


FIGURE 5.4: Intra-Set write variation (lower is better)

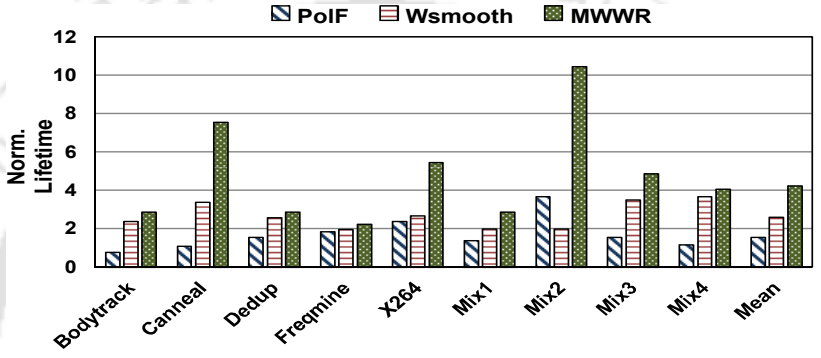


FIGURE 5.5: Normalized lifetime with respect to STT (higher is better)

λ. Note that hot-sub-way(s) represent the sub-way(s) having maximum write count(s) among the other sub-ways.

5.4.2 Relative Lifetime Improvement

Figure 5.5 shows the normalized lifetime with respect to STT and is presented against the STT, Polf, and Wsmooth. Our proposed technique MWWR improves the lifetime by 4.25 times against the STT, 2.71 times against the Polf and, 1.63 times against the Wsmooth. These respective improvements are basically due to the reduction in the write variation coefficient values by the proposed technique MWWR.

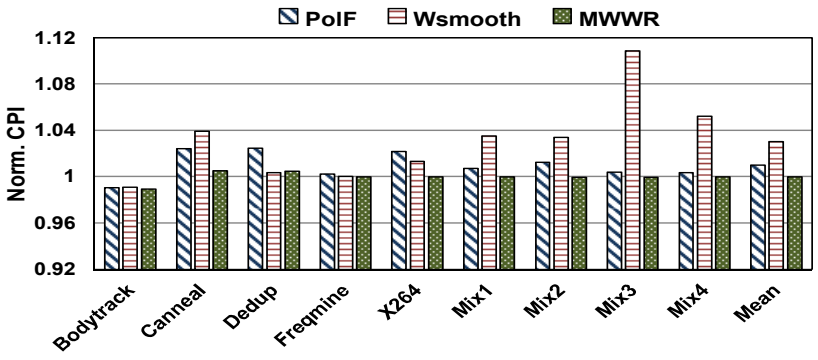


FIGURE 5.6: Normalized CPI with respect to STT (lower is better)

5.4.3 Effect on Performance

Figure 5.6 presents the normalized speedup (against the STT). MWWR maintains the same performance with the baseline STT. This is because MWWR evicts only LRU block from the other sub-ways, which in turn increase the miss rate only by 2.4%. The respective values in the increase of miss rate by PoLF and WSmooth are 12.8% and 29.3%. We observe 1% degradation in CPI with respect to STT by Polf due to increased allocations and evictions of MRU blocks in the cache. WSmooth shows performance degradation for STT by 3.32%. This degradation is due to turning off the sub-ways in each module of the cache that in turn increase the miss rate, and the extra cycles taken for the write operations due to block transfer from hot sub-way to cold sub-way inside the module. However, there is no impact on the performance loss compared to baseline STT despite the fact that the lesser cache availability at each module is due to the eviction of the LRU blocks from the other sub-ways (other than the write-restricted sub-way). Note that there is a trade-off between the number of each sub-way available for the allocation at each module and the performance loss, which can be easily seen from the table 5.5 (at row 4 and 5).

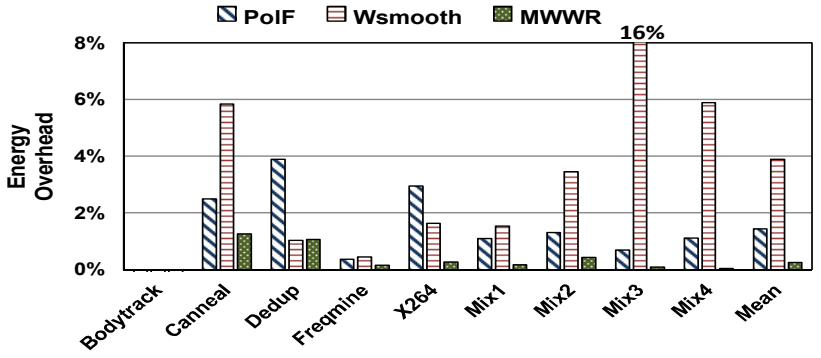


FIGURE 5.7: Energy overhead with respect to STT (lower is better)

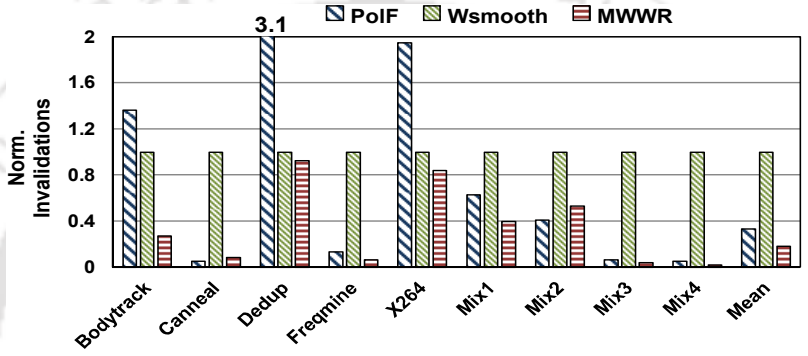


FIGURE 5.8: Normalized invalidation with respect to Wsmooth (lower is better)

5.4.4 Effect on Energy

Due to write redirections and the transfer of tags, MWWR consumes slightly more energy compared to baseline STT as shown in fig. 5.7. The energy overhead percentage against the baseline STT is merely 0.27%. However, with respect to PoLF and, Wsmooth, there is an improvement in the energy overhead in MWWR by 1.14% and, 3.3%. This gain in the energy is basically due to two reasons: (i) Polf invalidates MRU which in turn increases the allocations (writes) in the cache. (ii) Wsmooth moves the block from the hot sub-way to the cold sub-way which incurs extra writes.

Metric	IntraV	Lifetime	EDP Loss
MWWR	84%	3.10	0.32%

TABLE 5.4: Comparison analysis between FAWLT and MWWR

5.4.5 FLASH based Adaptive Wear Leveling Technique (FAWLT) versus MWWR

As same as in the previous chapter, the comparative analysis is illustrated between FLASH based Adaptive Wear Leveling (FAWLT) and the proposed wear leveling: MWWR. Table 5.4 list the analysis result between FAWLT [120] and MWWR. As can be seen from the table, the proposed approach works better than the FAWLT significantly, with a marginal increase in EDP.

5.4.6 Effect on Invalidation

Figure 5.8 shows normalized invalidations with respect to Wsmooth by the MWWR and Polf. MWWR reduces the invalidations by 45.3% and, 81.6% for Polf and, Wsmooth. Compared to Polf, the reduction is because of the selective invalidations by MWWR. The reduction compared to Wsmooth is mainly due to the difference between the write redirection policy of MWWR (transfer to any position of cache set) and block-transferring policy with Wsmooth (transfers from hot to one cold sub-way).

5.4.7 Storage Overhead

In our technique MWWR, we use A number of 10-bit C_{ij} counters with each module of the cache to record the write accesses of each sub-way. Besides, 128 42-bit swap buffers are used to transfer the tags. Also, the list $ModulewayList$ of size $m * M$ is composed by the entry of size $\log_2 A$. Thus, the percentage overhead implementation of MWWR compared to baseline STT-RAM is merely 0.02% for the selected values. On the other hand, the percentage savings in storage overhead

Param.	Norm. Lft.	IntraV Base	IntraV MWWR	Norm. EDP	Invalidations (k)
Reference	4.25	154.4%	61.4%	1.02	645k
I=2M	4.83	154.4%	57.6%	1.03	680k
I=10M	3.85	154.4%	64.5%	1.02	573k
m=8	4.94	154.4%	60.3%	1.04	786k
m=2	3.86	154.4%	67.4%	1.01	458k
M=256	4.68	154.4%	59.4%	1.03	658k
M=64	3.9	154.4%	62.8%	1.02	612k
8MB	4.92	119.3%	41.2%	1.02	751k
32MB	4.1	186.5%	85.7%	1.02	504k
A=8way	3.70	120.9%	57.4%	1.04	714k
A=32way	5.47	189.4%	79.7%	1.02	480k

TABLE 5.5: Comparative analysis for different parameters of L2 cache and algorithm (Lft.= lifetime, Base = baseline STT-RAM) ref = 16MB, 16 way, m = 4, M = 128, I = 5M

of MWWR compared to Wsmooth is 81.1% (Note that the storage bits required for counters and swap buffer are taken from [33]).

5.5 Comparative Analysis for Parameters

In addition to the results presented in the previous text, we also conducted experiments with different values of the parameters using various configurations of caches. Table 5.5 presents the comparative analysis where each row of the table shows the one change in parameter value compared to the reference value. Note that the value given in the table is with respect to STT.

5.5.1 Change in Interval (I)

With the larger interval value, the frequency of write-restricted sub-way selection process is reduced compared to the reference case. This, in turn, reduces the premature invalidation from the L2 cache because the blocks invalidated from the write redirection are mostly the LRU blocks. Further, with large interval value, the coefficient of intra-set write variation is larger with lesser improvement in a lifetime compared to the reference case. The opposite is seen in the case of the small interval.

5.5.2 Change in Write-Restricted Sub-Ways (m)

On increasing the write restricted sub-ways, more writes are redirected which in turn improves the lifetime and reduce the coefficient of intra-set write variation more than the reference case, and vice-versa, in case of a smaller number of write-restricted sub-ways.

5.5.3 Change in Number of Modules (M)

On increasing the number of modules, the granularity of wear-leveling is increased as it includes the smaller number of sets inside the module. This, in turn, improves the lifetime and reduces the intra-set write variation more than the reference case. The opposite case is seen with a smaller number of modules.

5.5.4 Change in Capacity

Cache with larger size experiences lesser capacity misses as compared to cache with a smaller size. This leads to the more extended residency of the block and higher write-variation compared to the reference case. In both cases: large and small-sized cache, our technique effectively reduces the write variation and improves the lifetime.

5.5.5 Change in Associativity (A)

Higher associative cache experiences less conflict misses compared to lower associative cache. This, in turn, increases the residency of a block and intra-set write variation inside the cache. In both the cases, higher and lower associativity, MWWR significantly reduces the intra-set write variation and improves the lifetime.

Cache Configuration	m	M	I
Small Size, Small Assoc.	↓	↓	= / ↓
Small Size, Large Assoc.	↑	↓	↑
Large Size, Small Assoc.	↓	↑	↓
Large Size, Large Assoc.	↑	↑	↓

TABLE 5.6: Recommended values of m, M and I

5.5.6 Recommended Values

Based on the above analyses, we present the best optimal trends of the proposed technique for the different configurations of caches in Table 5.6. Note that the recommended values are with respect to reference values (mentioned in Section 5.4) and = given in the table refers to the same value as the reference.

5.6 Summary

This chapter presented a technique: MWWR to mitigate the intra-set write variation: i.e., write variation inside the cache set. MWWR partitions the cache logically into multiple equal-sized modules and treats different sub-ways of each module for the certain predefined interval as write-restricted. Once the interval ends, the next set of sub-ways are chosen from the module, and the process continues until the end of execution.

The efficacy of the proposed technique is examined by comparing with the existing methods: Polf and WriteSmoothing and baseline STT-RAM. Experimental results show that the MWWR significantly reduces the coefficient of intra-set write variation and improves the lifetime by 4.25 times over baseline STT-RAM. Thus, reducing write variation by modular management of portions of the cache can increase their lifetime and make them as a capable candidates in the memory hierarchy.

भारतीय प्रौद्योगिकी संस्थान गुवाहाटी
Indian Institute of Technology Guwahati

Chapter 6

Inter-Set Wear Leveling using Dynamic Associativity Management Techniques

The previous chapter reported the wear-leveling techniques to mitigate the intra-set write variation. This chapter presents two efficient methods for lifetime longevity of NVM cache by reducing the inter-set write variation. Both the strategies illustrated in this chapter are using the concept of Dynamic Associativity Management to minimize the write variation across the cache set. The proposed policies partitions the cache sets into groups called fellow groups. Every cache set has two logical parts: Normal and Reserved. Cache sets within a fellow group can use the static/dynamic reserved parts from their fellow sets to distribute the writes uniformly. To measure the efficacy, the proposed strategies are evaluated with the baseline and the existing method in the quad-core system.

6.1 Introduction

With the limited write endurance, the lifetime of NVM cache is further affected by the write variations generated by the applications running on multiple cores.

Because of the write variations, the running applications create certain hot-spots in a cache to better utilize the temporal locality. In a cache, the write variations are categorized into two types: *Inter-set* and *Intra-set* write variations. This chapter present an efficient techniques to reduce the inter set write variation.

Our first technique: Fellow Set with Static Reserve Part (FSSRP) logically partitions the cache sets into groups of cache sets called fellow groups. Every group has two logical parts: Normal Part and Reserve Part (NP and RP). The working of NP in the group is the same as conventional cache. While the RP of the group is used to handle the non-uniform write distribution of the heavily written sets in the group. In other words, a cache sets within a fellow group can use the reserve parts from their fellow sets to distribute the writes uniformly. However, the major hurdle with the employment of the FSSRP is the limited lifetime improvement due to large number of write redirections in the static or fixed RP section of the cache which in turn contributes to the intra-set write variation. To overcome this, our second technique: Fellow Set with Dynamic Reserve Part (FSDRP) based on FSSRP, logically divides the cache into multiple equal-sized windows. During the execution, a different window of the cache is used as the RP section for certain predefined interval in order to distribute the writes. By this way, the FSDRP disperses the redirected writes over the cache.

We implemented our proposed schemes on STT-RAM based non-volatile cache. However, the techniques can be easily extended to other non-volatile caches such as PCRAM and ReRAM based caches. The main contributions of this chapter are as follows:

- We present the efficient techniques to reduce the inter-set write variation that helps to improve the lifetime of non-volatile caches.
- Our first technique: Fellow Set with Static Reserve Part (FSSRP) partitions the cache sets into groups called fellow groups. Each cache set in the group has two logical parts: Normal and Reserve. Sets within the fellow group can use the reserve parts from their fellow sets to distribute the writes uniformly.

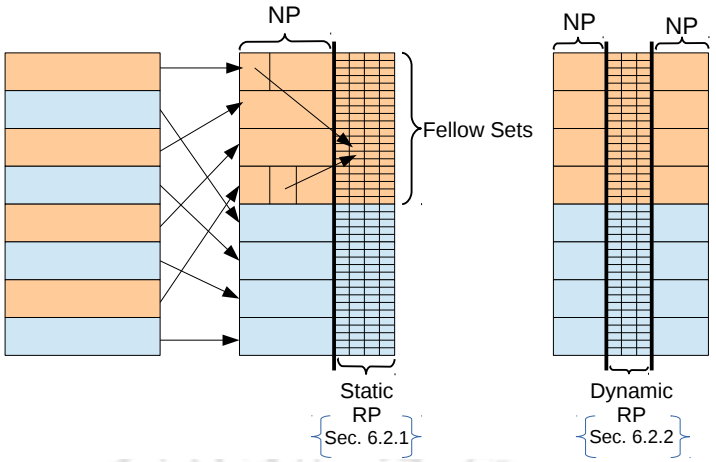


FIGURE 6.1: General overview of contribution in chapter 6

- Our second technique: Fellow set with Dynamic Reserve Part (FSDRP) is based on FSSRP in terms of the fellow group. In addition, it further partitions the cache vertically into multiple equal-sized windows. During execution, a different window of the cache is used as reserve part over a specified interval to distribute the writes uniformly.

- We use full system simulator GEM-5 [118] for experimental evaluation. Results are compared with the existing technique: Swap Shift [70] and the baseline STT-RAM-based cache with no support of wear leveling. We also provide a detailed analysis with different configurations of the LLC and, by varying the parameters of the techniques.

Figure 6.1 presents the general overview of the proposed contribution in this chapter.

The rest of the chapter is organized as follows: Section 6.2 present the proposed wear leveling techniques: FSSRP and FSDRP. Experimental Setup is discussed in section 6.3. Results and analysis are illustrated in section 6.4. The analysis with various parameters of the proposed techniques with the different configuration of caches are reported in section 6.5. Finally, section 6.6 summarize the chapter.

6.2 Proposed Wear Leveling Techniques

In this section, we will illustrate both of our proposed techniques: (i) Fellow Sets with Static Reserve Part and (ii) Fellow Sets with Dynamic Reserve Part.

6.2.1 Fellow Sets with Static Reserve Part (FSSRP)

6.2.1.1 Architecture

The main idea behind our proposal: FSSRP is exploiting the Dynamic Associativity Management (DAM) technique towards the wear leveling. Our technique partitions the cache into a group of sets called fellow groups. To enable DAM for a cache set, we partition the cache set into two parts: Normal (NP) and Reserve (RP) part. The working of NP part of the cache set is the same as the conventional cache. While the RP part of all sets in a fellow group can be used by all the sets within the group. In other words, if a set has high write usage, then it can use the space from the RP section of its fellow group to store its block. This way, the associativity can be dynamically managed. In our proposal, we do not intend to increase associativity but to use the feature of RP and fellow sets towards wear leveling. In particular, for a set that has a large number of writes taking place, such a set can avoid additional writes to itself by redirecting the writes to RP section of a lightly written set in the fellow group. Using set level write counters, one can decide the write redirection.

The issue remains to search these relocated blocks from its home set to other sets in its fellow group. For this, an additional mapping table: TaG Storage (TGS) is used. The entry in the TGS consists of a valid bit and the tag address of the block present in RP part. Note that each entry of TGS has one to one mapping with each entry of RP part.

Consider a Set Associative Cache having S number of sets and associativity A. The size of each group in the cache is m, and in each set of the group, r number of

the ways are reserved for RP. Note that the cache sets in the group are statically mapped. The group in the cache has the following characteristics:

- The distance between the two sets of the fellow group in the cache: $S/m - 1$
- Total number of groups in the cache: S/m.

The structure of TGS has the following attributes:

- Number of sets in TGS: $S_{tgs} = S/m$
- TGS Associativity: $A_{tgs} = r * m$
- Number of Blocks in TGS: $B_{tgs} = r * S$

Each block of RP in the cache has one to one mapping with each entry of the TGS. For a given TGS set number (S_{tgs_i}) and TGS way number (A_{tgs_j}), the cache set (S_k) and cache way (A_l) can easily find out with the help of following equations.

$$A_l = (A - r) + (A_{tgs_j} \% r) \tag{6.1}$$

$$S_k = ((A_{tgs_j}/r) * S_{tgs}) + S_{tgs_i} \tag{6.2}$$

Similarly, for a given cache associativity (A_v) and cache set (S_u), the respective TGS set (S_{tgs_n}) and TGS way (A_{tgs_m}) can be simply mapped with the help of following equation:

$$A_{tgs_m} = (S_u/S_{tgs}) * r + (A_v - (A - r)), \quad (A - r) \leq A_v \leq A \tag{6.3}$$

$$S_{tgs_n} = S_u \% S_{tgs} \tag{6.4}$$

In addition, with each set in the cache, the write counter is associated. The write counter is used to count the number of writes in the cache set. Further, with each

block in NP, a write bit is associated. The use of write bit and the write counter is explained in the next subsection.

The example of our architecture is presented in figure 6.2. For this example, consider a 16-way associative ($A = 16$) L2 cache having 8 sets ($S = 8$), partitioned into two parts: NP and RP. The number of ways in NP and RP is 12 and 4 ($r = 4$) respectively. Let the fellow group size be set to, $m = 2$. For the given values, the total number of the group to be formed in the cache is $S/m = 4$, and the distance between the two sets in the group is $S/m - 1 = 3$. In our example, these four groups are labeled by G0, G1, G2 and G3, and the distance between the set 0 and set 4 belongs to the group G0 is 3. The structure in TGS has following attributes: $S_{tgs} = S/m = 4$, $A_{tgs} = r * m = 8$ and $B_{tgs} = r * S = 32$. Each entry in TGS corresponds to a fellow group. In our example, set-0 of TGS has entries for tags belongings to blocks in G0 i.e., way 12-15 of cache set-0 and set-4.

6.2.1.2 Operation

The operation of the proposed technique is elaborated through algorithm 5. In the algorithm, the tunable parameter I is used as the predefined interval (line 1). The write counter associated with each set is represented by the variable W_i (line 2). Similarly, the write bit incorporated with each block in the NP part of the cache is represented by b_{mn} (line 3). The lightly written set of the group is represented by S_l. The decision of the light written set (S_l) in the group is taken with the help of write counters associated with the sets in the group. In other words, set which has a low value of write counter (W_i) is treated as lightly written set (S_l) of the group.

For the initial I cycles, the cache is used as a normally available cache. During the interval I, if any write happens to any block or in any set, the corresponding write bit of the block b_{mn} is set and write counter W_m of the cache set is incremented (line 4 to 6).

Algorithm 5 FSSRP Wear Leveling Algorithm

I : Predefined interval.
W_i : Write counter associated with set i. $0 \leq i \leq S$
b_{ij} : Write bit associated with the block in set i and way j. $0 \leq i \leq S$, $0 \leq j < (A - r)$
Run application for I cycles treating the cache as a normal cache.
During I cycle, the write counter (W_i) is incremented with each write in set i.
Similarly, the write bit(b_{ij}) is set with each write in the block.
repeat
  for each request R coming from L1 cache to block B in L2 cache **do**
    if $R = ReadHit$ **then**
      The Read operation is performed on the block B irrespective of its location.
    else if $R = WriteHit$ **then**
      if Block B is found in NP part of cache **then**
        if the write bit b_{ij} of the block B is set **then**
          if there exist a light written set S_l in the group **then**
            The write request for the block B is redirected to the location L in the RP part of S_l. ▷ **Block B moved to RP on first write back**
          else
            The write operation is performed on Block B. Increment the write counter (W_i) and keep the write bit set.
          end if
        else
          The write operation is performed on Block B. Increment the write counter (W_i) and set the write bit b_{ij}.
        end if
      else
        The write operation is performed on block B. Increment the write counter of the set in which the write operation is performed. ▷ **Block B in RP part**
      end if
    else
      Forward the Request R to main memory. Keep the newly arrived block in NP part of cache. ▷ **cache miss**
    end if
  end for
until the end of the execution

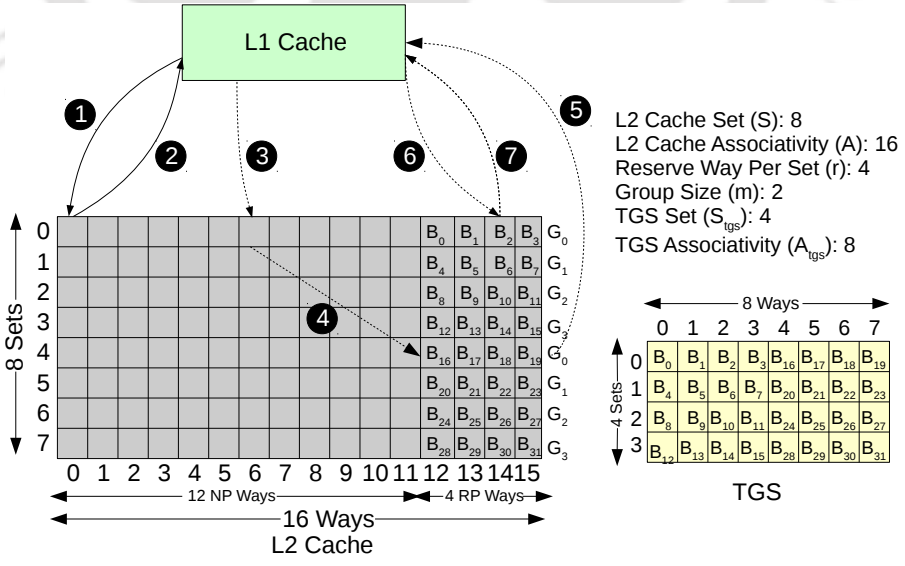


FIGURE 6.2: Working example of FSSRP wear leveling policy

For the request R coming from L1 to L2 cache, the tag lookup operation is performed in NP part of the cache. Simultaneously, the tag of the requested block is also searched in the RP part of the group with the help of TGS. Note that the TGS set location is found out with the help of equation 6.4. If the requested block is present in the NP part of the cache, then it is a direct hit. Otherwise, it is an indirect hit. Note that if the requested block B is in RP part of the cache, then the cache set and cache way of the block is found out with the help of equation 6.2 and equation 6.1.

Depending upon the result of the lookup operation, different operations are performed in the cache, which can be explained as follows:

- ***Read Hit:*** For a read request R, if the block B is present in the cache. The read operation is performed on the block irrespective of its location (either NP or RP) (line 9 to 10).
- ***Write Hit (PUTX or write-back) and block B in NP part of the cache:*** If the requested block B is present in the L2 cache with write bit set, and if there exist any lightly written set (S_l) in the group, the write request is redirected from the current set to the RP part of S_l. If there is an invalid line(s) in RP of the S_l, the request R is redirected to the first invalid line and the block B will be invalidated from the NP part of the cache. Otherwise, if there is no invalid line, then the LRU victim line is selected from the RP of S_l. In this case, the write-back operation of the victim line is scheduled to next level of memory. Afterward, the write request is redirected to the generated location, and the block will be invalidated from the NP part of the cache. Subsequently, the tag entry of the redirected block is created in the TGS with the help of equation 6.4 and equation 6.3 (line 14 to 16). On the other hand, if no lightly written set exists in the group or the write bit (b_{mn}) of the block is not set. In these cases, the write request is performed in its current location, and the bit b_{mn} of the block is set (line 17 to 21). Note that when there is no lightly written set in the group, this implies that the current set itself is the lightly written set

of the group. Once the request is served, the write counter (W_m) of the cache set in which the write is performed is incremented.

- ***Write Hit (PUTX or write-back) and block*** B ***in RP part of the cache:*** If the requested block B is present in L2 cache and it belongs to RP part of the cache (indirect hit), the write request is performed normally on the block B. Afterward, the write counter of the set (W_i) in which the write is performed is incremented (line 22 to 24).

- ***Cache Miss:*** If the requested block is not present in the L2 cache, the request R from the L1 cache will be forwarded to the next level of memory. In this case, the newly arrived block will be placed in the NP part of the cache (line 25 to 27).

The working methodology of an algorithm is presented in fig. 6.2. Note that the details about the structure of the cache and the TGS are already explained in section 6.2.1.1.

Example: To demonstrate the method, three cases are considered with respect to set-0. In the first case, a read request from L1 cache to way-i of L2 cache (shown by the arrow with label-1) is served normally (as represented by arrow-2) irrespective of the location (NP or RP) of the block. In the second case, a write request from L1 cache to block in way-6 of L2 cache (shown by arrow 3) which implies the NP part of the cache. In this case, the write request is redirected to the RP part of lightly written set in the group, say set 4 in our example (shown by arrow 4). Once the write operation is performed in set 4 and the entry in TGS is updated in set 0, the write-back acknowledgment is sent back to the L1 cache (as represented by arrow 5). In the last case, the write hit in the way-14 of L2 cache i.e., RP part of the cache (shown by arrow 6) is served normally. Once the operation is performed, the write-back acknowledgment is sent back to the L1 cache (shown by arrow 7).

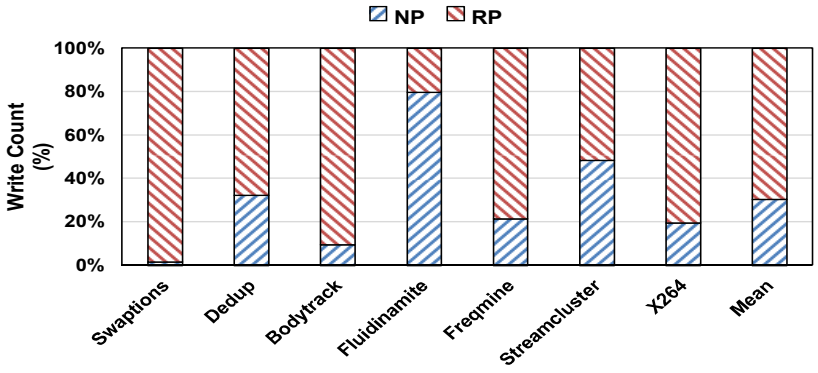


FIGURE 6.3: Write count percentages in the different section of FSSRP

Workloads	Swap	Dedup	Body	Fluid	Freq	Stream	X264	Mean
IntraV %	1.3%	45.8%	47.1%	-2.5%	7.1%	8.7%	1.2%	15.5%

TABLE 6.1: Percentage increase in coefficient of Intra-Set write variation

6.2.1.3 Limitation

The limitation of FSSRP is the extensive write accesses and write redirections in the static limited sized RP section of the cache. Figure 6.3 presents the write count percentages in the NP and RP sections of the cache for different benchmark applications (Details about the experimental setup over the selected value is given in section 6.3). The conclusion that can be derived from fig. 6.3 is that, on an average, 69.6% of the write access is handled by the static limited sized RP section of the cache which in turn generates the intra-set write variation as shown in the table 6.1. This increment in the coefficient of intra-set write variation by 15.5% over the baseline limits the lifetime improvement (calculated from eq. (2.4)) despite the reduction in inter-set write variation by the proposed approach: FSSRP. All these facts motivate us to make RP section of the cache dynamic in an attempt to control the intra-set write variation.

6.2.2 Fellow Sets with Dynamic Reserve Part (FSDRP)

6.2.2.1 Architecture

The architecture framework of FSDRP is based on FSSRP in that terms it also creates the fellow groups. The main idea of FSDRP is to further logically partition the cache vertically into multiple uniformly sized windows (such that each window contains an equal number of ways) and over a certain period of execution, a single-window is used as an RP section, exclusively. In other words, instead of partitioning the cache into two static parts: NP and RP, FSDRP partitions the cache into multiple equal-sized windows of size r ways and dynamically use a different partition as RP for a certain period over the execution. However, during that time, the remaining ways behave as the NP section. The benefit obtained by using different windows as RP over the execution is that the writes are not concentrated on one part of the cache but get dispersed over the different ways of the cache set in the fellow group. Note that here in FSDRP, our intention is not to reduce the intra-set write variation of the cache, but to distributes the redirected writes of the RP section in the cache uniformly over the set.

The one to one static mapping between the TGS entry and relocatable blocks in the other sets of the fellow group has evolved as a significant bottleneck towards the implementation of FSDRP. Practically, the dispersion of relocatable blocks in the fellow sets by the dynamic RP window destroys the static mapping setup by FSSRP. To deal with this, we add a field called *win_num* with each entry of TGS and a relocate bit (r_{ij}) with each entry of the cache along with the write bit (b_{ij}). The use of the *win_num* field is to store the partition or window number where the relocatable block resides in the cache. The use of relocate bit (r_{ij}) is to identify the normal block from the relocated block in the cache set since the RP window keeps moving.

The partition or window has the following characteristics:

- Size of the window or partition in the cache: r.

- Total number of window or partition in the cache (P): A/r.
- Partition or window number for a given way number (W) in the cache: $\lfloor W/r \rfloor$

Similar to FSSRP, for a given TGS set (S_{tgs_i}) and TGS associativity (A_{tgs_j}), the corresponding cache set (S_k) can be easily mapped with Equation (6.2). However, the corresponding cache way (A_l) is identified by the following search operation in the cache:

$$A_l = Search(TGS[S_{tgs_i}][A_{tgs_j}].win\_num, TGS[S_{tgs_i}][A_{tgs_j}].tag) \tag{6.5}$$

The $Search()$ used in eq. (6.5) takes two arguments: win_num and tag address (tag) from the respective location of TGS and searches the corresponding block in the cache.
Similarly, the respective TGS set (S_{tgs_n}) for the cache set (S_u) and cache way (A_v) is mapped from eq. (6.4). Whereas, the TGS way (A_{tgs_m}) is derived by using the following search operation:

$$A_{tgs_m} = Search(Cache[S_u][A_v].tag, A_{tgs_{st}}, A_{tgs_{st}}+r), \quad A_{tgs_{st}} = (S_u/S_{tgs})*r \tag{6.6}$$

Here, the $Search()$ used in eq. (6.6) takes three arguments: tag address of the cache block ($Cache[S_u][A_v].tag$) and the range of the TGS way location as second and third argument ($A_{tgs_{st}}$ to $A_{tgs_{st}} + r$) where the searching takes place for the respective cache block.

Example: The example of FSDRP architecture is depicted in Figure 6.4. In the example, a 16-way ($A = 16$) associative L2 cache having 8 sets is considered for the demonstration with the values of $m = 2$ and $r = 4$. With these given parameters, the cache is partitioned into four ($A/r = 4$) equal-sized windows of size four ways each ($r = 4$). As shown in the example, these four windows are labeled with Win_0, Win_1, Win_2 and Win_3 and the relocatable blocks from the different cache sets of the fellow groups are dispersed in these four different windows. The lookup of these relocatable blocks is performed with the help of TGS through eqs. (6.2) and (6.5). As shown in the example, the block B_{12} placed in window W_0 of the

cache has an entry in TGS, and it is searched with the help of *win_num* field (value 0) stored within the TGS entry. As can be seen from the figure, we have added relocate bit to distinguish the relocatable block and write bit for write redirection with every block of the cache. For example, the write bit and the relocate bit for the relocatable block B_{31} are set to 0 and 1.

6.2.2.2 Operation

The operation of the FSDRP is elaborated through Algorithm 6. In this algorithm, the use of parameters I, W_i and b_{ij} is same as the FSSRP (line 1 to 3). In addition to these parameters, the parameter P acts as the total number of logical partitions or windows in the cache (line 4). The relocate bit associated with each block of the cache is represented by variable r_{ij} (line 5).

For the initial I cycles of the process execution, the cache is available as a normal cache. Meanwhile, during the interval (I), the write operation perform to any block in the cache set increments the write counter (W_i) of that cache set, and the respective write bit (b_{ij}) of the block is set (line 6 to 8).

Once the application crosses first I cycles, one window of the cache is selected and treated as RP section of the cache and the rest of the windows act as an NP section. Afterward, periodically for every interval I, a new window is treated as RP by rotation (line 9 to 14). The process continues until the execution is over.

In the meantime, between the intervals, for each request R coming from L1 cache to L2 cache, the tag lookup operation is performed in the L2 cache simultaneously in both NP and RP (through TGS by eq. (6.4)). Note that, in the case of an indirect hit, the respective cache set and the cache way is mapped through eqs. (6.2) and (6.5) (line 15). Depending upon the result of the lookup and the cache request, different operations are performed in the L2 cache:

- ***Read Hit:***The read operation is same as the FSSRP, as given in section 6.2.1.2 (line 16 and 17).

Algorithm 6 FSDRP Wear Leveling Algorithm

I : Predefined interval.
W_i : Write counter associated with set i. $0 \leq i \leq S$
b_{ij} : Write bit associated with each cache block in set i and way j. $0 \leq i < S$, $0 \leq j < A$
P : Number of logical partition or windows.
r_{ij} : Relocate bit associated with each cache block in set i and way j. $0 \leq i < S$, $0 \leq j < A$
Run application for I cycles treating the whole cache as a normal available cache.
During I cycle, the write counter (W_i) is incremented with each write in set i.
Similarly, the write bit(b_{ij}) is set with each write in the block.
After I cycle, treat one window of the cache as a RP window and rotate window number in a round robin fashion.
repeat
  for every interval I **do**
    $i = (i+1)\%P$
    Window Win_i is selected as a RP section for the current interval I.
    Windows other than the W_i is treated as NP section of the cache.
    for each request R coming from L1 cache to block B in L2 cache **do**
      if $R = ReadHit$ **then**
        The Read operation is performed on the block B irrespective of its location.
      else if $R = WriteHit$ **then**
        if Block B is found in NP part of cache **then**
          if the write bit b_{ij} of the block B is set and the relocate bit r_{ij} is not set **then**
            if there is any light written set S_l exist in the group **then**
              The write request for the block B is redirected to the location L in the RP part of S_l. ▷ **Block B move to RP after first write back**
              The relocate bit (r_{lm}) for the redirected block is set.
            else
              The write operation is performed on Block B. Increment the write counter (W_i) and keep the write bit set.
            end if
          else
            The write operation is performed on Block B. Increment the write counter (W_i) and set the write bit in set i and way j.
          end if
        else
          The write operation is performed on block B. Increment the write counter of the set in which the write operation is performed. ▷ **Block B in RP part**
          if r_{ij} is not set **then**
            Set the write bit (b_{ij}) for the block B.
          end if
        end if
      else
        Forward the Request R to main memory. Keep the newly arrived block in NP part of cache (location other than Win_i). ▷ **cache miss**
      end if
    end for
  end for
until the end of the execution

- ***Write Hit (PUTX or write-back) and block B in NP section of the cache:*** In case of a write request R, if the requested block B belongs to the NP section of the cache then we have two cases:

  – In the first case, for the requested block B, if the write bit (b_{ij}) is set and the relocate bit (r_{ij}) is zero. And, at the same time, if the lightly written set (other than the current set) (S_l) is present in the fellow group, the write request R is redirected to the RP window (Win_i) of the S_l. In this case, if the invalid entries exist in the TGS (within the range $A_{tgs_{st}}$ to $A_{tgs_{st}} + r$) and the RP window, the write request (R) is simply redirected to the invalid location

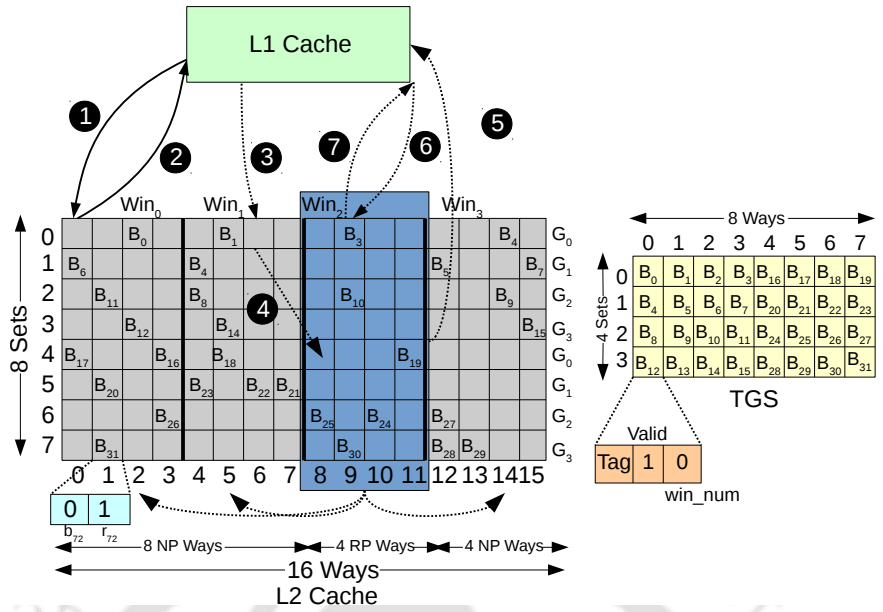


FIGURE 6.4: Working example of FSDRP wear leveling policy

of the RP section by updating the TGS entry. On the other hand, if there is no invalid entry present in the RP window of the cache or if there is no vacant entry present in the TGS, the respective LRU victim entry is picked from the RP window or from the TGS and the write-back operation is performed for either or both the entries of the L2 cache. Note that, the cache location of the LRU TGS entry is mapped by the eqs. (6.2) and (6.5). Afterward, the write request R from an L1 cache is redirected to the newly generated entry in the RP window. Simultaneously, the newly generated TGS entry is also updated with the redirected data entry attributes. Once the write request R is redirected, the subsequent block B is invalidated from the NP section and the relocate bit (r_{lm}) is set for the redirected block in the RP window section for future identification (line 21 to 23).

– In the second case, if the write bit (b_{ij}) is not set or if the relocate bit (r_{ij}) is set or if there is no lightly written set other than the current set exist in the group, the write operation is performed on the current location of the block B by setting the write bit b_{ij} (line 24 to 28). Note that the setting of the relocate bit r_{ij} implies that the current block belongs to the other set of the fellow group.

Once the write operation is performed, the respective write counter (W_i) of the cache set in which the write operation is performed is incremented.

- ***Write Hit (PUTX or write-back) and block*** B ***in RP section of the cache:*** In this case, the write request R is performed normally from the current location of the block B. If the relocate bit (r_{ij}) of the block, B is not set then the corresponding write bit (b_{ij}) is set. This implies that the block B is not yet redirected and currently belongs to the home set (line 30 to 33).

- ***Cache Miss:*** In case of a cache miss, the request R from L1 cache is forwarded to the next level of memory (main memory in our case). The incoming block from the main memory is placed into the window other than the RP window (Win_i). In particular, the block is situated in one of that window which is currently treated as NP section of the cache. The write bit (b_{ij}) and the relocate bit (r_{ij}) of the cache location in which the block is placed are reset (line 36 to 38).

Example: The working example of FSDRP is explained through fig. 6.4. In the example, the window Win_2 is treated as an RP section, and the rest of the windows (Win_0, Win_1, Win_3) act as an NP section of the cache. As same as the FSSRP, three cases are considered to demonstrate the method concerning set-0. In the first case, a read request (arrow 1) is normally served irrespective of the location (NP or RP) (arrow 2). In the second case, a write request (arrow 3) to the block (way-6) that belong to the NP section is redirected (arrow 4) to the RP section window (Win_2) of the lightly written set (set-4 in our case) of the fellow group. At the same time, the respective TGS entry of set-0 is updated with the individual data attributes. Once the write operation is performed, the write-back acknowledgment is sent back to the L1 cache (arrow 5). In the last case, the write request (arrow 6) to the block B_3 in the RP section of the cache is served normally by the L2 cache with the write-back acknowledgment (arrow 7).

Components	Parameters
Processor	2Ghz, Quad Core, X86
L1 Cache	Private, 32 KB SRAM Split I/D caches, 4-way set associative cache, 64B block, 1-cycle latency, LRU, write-back policy
L2 Cache	Shared, 64B block, LRU, write-back policy
Protocol	MESI CMP Directory

TABLE 6.2: System parameters

	Leakage Power (mW)	Hit Energy (nJ)	Miss Energy (nJ)	Write Energy (nJ)	Hit Latency (ns)	Miss Latency (ns)	Write Latency (ns)
L2 Cache Configurations Attributes							
16MB, 16way	15.674	0.367	0.096	4.322	78.453	11.854	271.035
8MB, 32way	8.116	0.366	0.185	6.454	74.792	8.259	270.981
8MB, 16way	8.030	0.273	0.093	4.387	78.497	11.964	270.981
8MB, 8way	7.983	0.227	0.047	3.221	74.454	7.921	270.981
4MB, 16way	7.960	0.217	0.093	4.228	23.876	5.575	126.585
TGS Configurations Attributes							
1024 Set, 16way	8.120	0.018	0.001	0.018	1.349	0.005	1.349
1024 Set, 32way	15.96	0.034	0.002	0.034	1.66	0.010	1.66
2048 Set, 8way	8.122	0.018	0.001	0.018	1.352	0.005	1.352
2048 Set, 16way	15.96	0.033	0.002	0.033	1.673	0.007	1.673
2048 Set, 32way	30.51	0.067	0.003	0.067	3.58	0.012	3.58
4096 Set, 8way	15.96	0.034	0.002	0.034	1.67	0.007	1.67
4096 Set, 16 way	30.51	0.067	0.003	0.067	3.58	0.010	3.58

TABLE 6.3: Timing and energy parameters for STT-RAM L2 cache and SRAM based TGS

6.3 Experimental Setup

We implemented our proposed schemes on a full system simulator GEM-5 [118]. Table 6.2 shows the system parameters used in our simulations. We perform the experiments on a quad-core system with the different configuration of L2 or LLC. Table 6.3 reports the timing and energy parameters for these configurations. The timing and the energy parameters are obtained by using NVSIM [21] at 32 nm technology node.

We compared our proposed techniques with baseline STT-RAM cache that uses LRU as a replacement policy with no wear leveling strategy associated and, the existing method: Swap Shift. In our experiment, the value of *SwapTh* is set to 511. The rationale behind the large value of the *SwapTh* is to restrict the frequent invalidation process of data blocks in the cache set. As the frequent invalidation process increases the accesses in the main memory that results in the extra performance and energy overhead.

In the proposed schemes, the searching of the block in the home set and the other sets of the fellow group (through TGS) will take place in parallel (as it can be easily seen from the table 6.3, the latency of TGS can be easily overlapped with the cache access latency), so it does not affect the system performance. However, the write redirection of the blocks from home set to the other set takes three extra cycles and an additional swap buffer to the transfer the tag. These three cycles are divided as follows: 1 cycle for tag transfer into the swap buffer, one cycle for writing the tag in swap buffer and, one cycle for transferring the tag to TGS and RP part of the cache. In addition to these cycles, an extra cycle is required in FSDRP for the searching of the block into the respective window of the other sets in the fellow group. We have considered all these overheads in our simulations.We also take into account the energy consumption due to accesses in the TGS and the area overhead of extra hardware circuitry compares to the main STT-RAM-based data array (refer section 6.4.7). The timing, energy (as shown in the table 6.3) and area overhead of TGS (made up of SRAM) is modeled by using NVSIM. Note that we have considered the Fast access mode of the NVSIM during the modeling of TGS to make sure the parallel access. There is some logic overhead associated with the algorithms 5 and 6, which will consume a mere amount of extra area overhead.

We verified our proposed techniques with the help of PARSEC benchmarks suite [6]. From the suite, we have used seven applications (Swaptions (Swap), Dedup, Bodytrack (Body), Fluidanimate (Fluid), Freqmine (Freq), Streamcluster (Stream), X264) in the simulation. Note that the detailed description of the simulation framework is given at appendix A.

6.4 Results and Analysis

We evaluate our proposed approaches on a quad-core system. Out of the different configurations of L2 cache, we conducted our experiment on 8MB, 16-way associative L2 cache. In the proposed schemes, we set the value for m, r, and

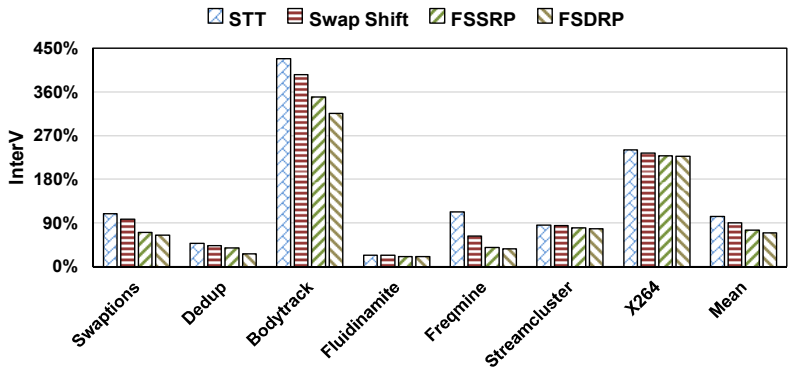


FIGURE 6.5: Inter-Set write variation of proposed schemes: FSSRP and FSDRP and, Swap Shift against baseline STT-RAM (lower is better)

I to 4, 4, and 5 million cycles. Later in the section, we analyze the effects by changing these values. We present our results on the following metrics: reduction in coefficient of Inter-set write variation ($InterV$) calculated with the help of eq. (2.1), percentage reduction in coefficient of Intra-set write variation over the proposed scheme: FSSRP and the baseline STT-RAM (computed with the help of eq. (2.2)), lifetime improvement percentage calculated with the help of eq. (2.4), speedup, energy overhead and the number of invalidation/flushes. In this chapter, the comparison analysis of proposed techniques with flash based wear leveling techniques is not presented. This is because flash-based wear-leveling aimed to reduce the unwanted erasure count between the blocks (consist of multiple pages) that leads to Intra-Block wear leveling. Whereas the proposals presented in this chapter are aimed to reduce the write variation across the cache sets in a bank i.e., Inter-Set write variation. Thus, it is unfair to compare the proposed techniques with existing flash-based wear-leveling proposals.

6.4.1 Coefficient of Inter-Set Write Variation

Figure 6.5 shows the Coefficient of Inter-Set Write Variation. Our proposed schemes reduce the coefficient of inter-set write variation from 103.9% (STT), 91.4% (Swap Shift), 76.3% (FSSRP) to 69.9% (FSDRP). The reduction in the coefficient of inter-set write variation is due to uniform write distribution across

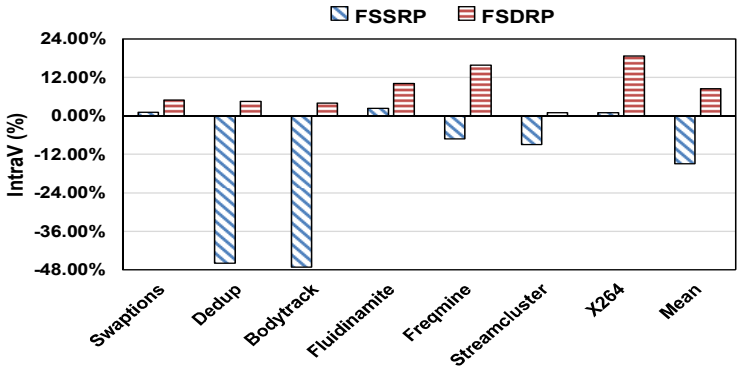


FIGURE 6.6: Percentage reduction in Intra-Set write variation by FSSRP and FSDRP over baseline STT-RAM (higher is better)

the cache sets by redirecting the writes from the heavily written sets to the lightly written sets of the fellow groups. However, further improvement in the inter-set write variation for FSDRP over FSSRP is due to different RP window throughout execution that restricts the write redirection of the some fraction of the blocks from their home sets. Note that, FSSRP redirects every block on the second write without considering its write intensity. On the other hand, this is not the case with FSDRP as it partially restricts the write redirection of the block that belongs to the current RP window.

6.4.2 Reduction in Coefficient of Intra-Set Write Variation

Figure 6.6 shows the percentage reduction in intra-set write variation by FSDRP. Compared to baseline and FSSRP, FSDRP reduces the intra-set write variation by 8.55% and 17.7%, respectively. The reduction in the intra-set write variation is mainly due to the dispersion of the relocatable blocks over the cache set by dynamic RP window. This reduces the possibility of write concentration in the specific section/region of the cache.

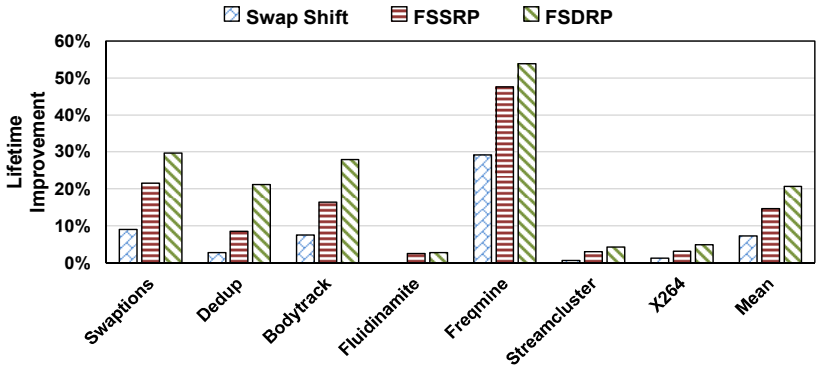


FIGURE 6.7: Lifetime improvement by FSSRP, FSDRP and, Swap Shift with respect to baseline STT-RAM (higher is better)

FIGURE 6.8: Normalized invalidations by the proposed techniques: FSSRP and FSDRP over Swap Shift. (lower is better)

6.4.3 Lifetime Improvement

Figure 6.7 presents the lifetime improvement percentage by the proposed schemes. Compared to STT-RAM, the improvement in the lifetime by FSSRP is 14.77% and by FSDRP is 20.77%. However, the respective values for the lifetime improvement over Swap Shift is 6.58% by FSSRP and 12.11% by FSDRP. These improvements by the proposed schemes are mainly due to the reduction in the coefficient of inter-set write variation. The development in the lifetime by FSDRP with respect to FSSRP is 3.03%. The reason for the further lifetime improvement by FSDRP is due to the reduction in intra-set write variation in lieu of the dynamic RP window.

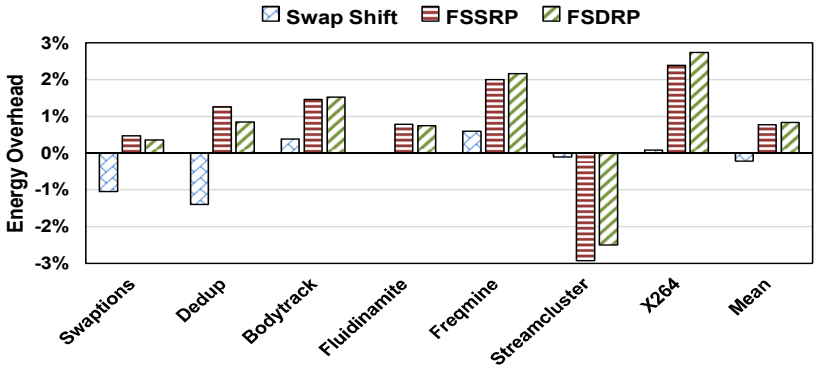


FIGURE 6.9: Energy overhead by the proposed techniques: FSSRP and FSDRP against baseline STT-RAM (lower is better)

6.4.4 Invalidation/flushes

The invalidation/flushes by the proposed schemes with respect to Swap Shift is depicted in fig. 6.8. Flushes take place due to the write redirection from heavily written cache sets to lightly written cache sets of the fellow group. Compared to swap shift, the proposed schemes reduce the invalidation by 21.59% (FSSRP) and 15.58% (FSDRP). However, as can be seen, the FSDRP increases the invalidation against the FSSRP by 7.66%. This is because, in FSDRP with each write redirection, two blocks gets evicted: (i) from current RP to accommodate the redirected block and (ii) in order to make an entry in the TGS one would need to evict the relocatable block from the current NP or RP section. Note that in some of the cases (dedup, fluid, freqmine, and x264), the number of invalidation by our techniques is large than the swap shift. This is because we set a more considerable value of threshold ($SwapTh$) in swap shift, to maintain the performance and energy consumption.

6.4.5 Energy Overhead

The energy overheads of the proposed schemes is shown in fig. 6.9. Note that the negative values in the figure imply energy savings. The energy overhead percentage over the baseline and the Swap Shift by the proposed schemes are 0.78% and 0.99% by FSSRP and, 0.84% and 1.05% by FSDRP. This marginal increment in

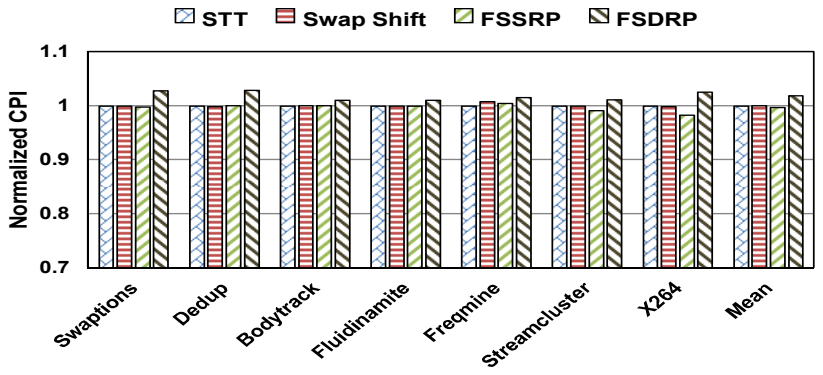


FIGURE 6.10: Normalized performance by the proposed techniques: FSSRP and FSDRP and Swap Shift with respect to baseline STT-RAM (lower is better)

the energy is basically due to the transfer of tag, invalidation/flushes, accesses in TGS and, the extra window search operation for the relocatable block in the cache by FSDRP.

6.4.6 Performance

The proposed scheme: FSSRP maintains the same performance as in the baseline STT-RAM and Swap Shift, as shown in Figure 6.10. Performance is not affected due to less available capacity in FSSRP due to two reasons: Firstly, searching for the block in RP section (with the help of TGS) takes place in parallel with the lookup operation in the NP section, and the writing in TGS will be in parallel with the writing in the RP. Secondly, due to the DAM, the set having higher write access will transfer its load to lightly written set in the fellow-group. However, a small degradation of 1% is observed in the CPI for FSDRP due to the time taken by extra search operations in the dynamic RP windows.

6.4.7 Storage and Area Overhead

In our proposed schemes, we use 12-bit write counter (W_i) to measure the write counts in the cache set. Besides this, we add a relocate bit (r_{ij}) and a write bit (b_{ij}) with each block in the cache. Further, each entry of TGS is made up of 42-bit

tag address (t), one valid bit (v) and $\log_2 A/r$ bits for window number (win_num) (used in case of FSDRP). We also add a 42-bit swap buffer to transfer the tag to the new location. Thus, the storage overhead of FSDRP and FSSRP are computed by using the following couple of equations:

$$O_{FSDRP} = \frac{S * W_i + S * A * (b_{ij} + r_{ij}) + S * r * (t + v + win\_num) + 42}{S * A * (B + t)} * 100 \tag{6.7}$$

$$O_{FSSRP} = \frac{S * W_i + S * (A - r) * b_{ij} + S * r * (t + v) + 42}{S * A * (B + t)} * 100 \tag{6.8}$$

In the above eqs. (6.7) and (6.8), B represents the block size. As an example, in our selected configuration: 8MB 16-way associative L2 cache with the following set of attributes: $m = 4$, $r = 4$ and $I = 5$ million cycles, the percentages of storage overhead in FSSRP and FSDRP are merely 2.21% and 2.52%.

Whereas, the area overhead of FSDRP and FSSRP with respect to baseline STT cache computed by using the NVSIM is 16%. Note that while obtaining the area parameters, we have taken into the consideration of parallel (or FAST access in the NVSIM) access with the STT-RAM main data cache array.

6.5 Parameter Comparison Analysis

In addition to the results presented in the previous section, we also performed experiments with different configurations of the cache and the parameters (m, r and I) for the algorithms. Here, we show the results for the reduction in inter-set write variation, the percentage reduction in the intra-set write variation over the baseline, lifetime improvement, EDP overhead, and the number of invalidations. This analysis is instrumental in picking the optimal values (of the parameters) for the proposed approaches in different cache configurations.

Param.	LI (%)	InterV Base	InterV FSDRP	IntraV Red. (%)	Norm. EDP	Invalid-ation(k)
Ref.($I = 5M$)	20.7%	103.9	69.9	8.5%	1.02	173k
$I = 2M$	23.2%	103.9	68.2	9%	1.03	188k
$I = 10M$	19.3%	103.9	73.3	3.5%	1.01	155k

TABLE 6.4: Comparison analysis for different interval values (I) (LI = Lifetime Improvement, Base = baseline STT-RAM) ref.= 8MB, 16-way, $m = 4$, $r = 4$ and $I = 5M$

Param.	Policy	LI (%)	InterV Base	InterV FS	IntraV Red. (%)	Norm. EDP	Invalid-ation(k)
Ref. (m=4)	FSSRP	14.7%	103.9	76.3	-14.8%	1.01	161k
	FSDRP	20.7%	103.9	69.9	8.5%	1.02	173k
m=8	FSSRP	15.5%	103.9	71.3	-17.6%	1.02	172k
	FSDRP	22.1%	103.9	68.9	2.2%	1.02	186k
m=2	FSSRP	12.8%	103.9	78.5	-13.6%	1.00	119k
	FSDRP	19.8%	103.9	73.5	9.3%	1.01	157k

TABLE 6.5: Comparison analysis for different fellow group size (m)

6.5.1 Change in Interval (I)

Table 6.4 reports the values over distinct intervals against the reference interval with $I = 5$ Million cycles in case of FSDRP. Change in the interval-span affects the RP window rotation process. Smaller interval increase the number of rotations of the RP window over the cache. Whereas, for the larger interval, the number of rotations of the RP window are reduced. Small interval value reduces the inter-set write variation more compared to the large interval. This is because the large interval value increases the residency of the block that belongs to the home set in the RP window. Such blocks residing in the RP window do not get redirected and thus increase the write count of the set and become eventually dead over the interval. In particular, the blocks belonging to the home set and the RP window incurs several writes before the RP window gets rotated to another location. Simultaneously, with large interval, the write concentration in the RP window region of the cache increases that in turn impacts (and increases) the intra-set write variation. However, in case of small interval values, due to increased invalidations and write redirections, the system performance is affected with more energy consumption, which further results into the increase in EDP.

Param.	Policy	LI (%)	InterV Base	InterV FS	IntraV Red. (%)	Norm. EDP	Invalid-ation(k)
Ref. ($r = 4$)	FSSRP	14.7%	103.9	76.3	-14.8%	1.01	161k
	FSDRP	20.7%	103.9	69.9	8.5%	1.02	173k
$r = 6$	FSSRP	12.7%	103.9	78.2	-18%	1.00	139k
$r = 8$	FSDRP	16.7%	103.9	73.8	2.1%	1.01	169k
$r = 2$	FSSRP	14.3%	103.9	75.8	-12.4%	1.02	201k
	FSDRP	19.3%	103.9	70.3	8.4%	1.03	187k

TABLE 6.6: Comparison analysis for different window size or RP size (r)

6.5.2 Change in Group Size (m)

Table 6.5 presents the results for the different group size (m) in the proposed schemes. Note that, the negative values in the table implies the increment in the intra-set write variation. Change in group size affects the availability of the lightly written sets in the fellow group. In particular, the large group size increases the chance of finding the lightly written set(s) in the group compared to small group size. With the large availability of lightly written set, the chances of write redirection increases, and this improves the lifetime and further reduce the inter-set write variation. However, due to increased invalidation and the write redirections, the system performance is marginally affected along with the energy consumption that in turn, increases the EDP. Also, the increased number of redirections populates more data in the RP window over the interval, which further impacts the reduction in intra-set write variation as can be seen by the reduced reduction percentage in the table 6.5 for both FSSRP and FSDRP.

6.5.3 Change in Window or RP Size (r)

Table 6.6 lists the result metrics for changing the window size (i.e. RP size (r)). Change in the window size affects the residency of the relocatable block in the different sets of the fellow group. An increment in window size increases the residency of the relocatable blocks. This, in turn, increases the write count of the lightly written set and it becomes heavily written set over the period. This reduces the possibility of finding the lightly written set in the group and leads to more inter and intra-set write variation compared to small RP/window size. However,

Param.	Policy	LI (%)	InterV Base	InterV FS	IntraV Red. (%)	Norm. EDP	Invalid-ation(k)
Ref. (C = 8MB)	FSSRP	14.7%	103.9	76.3	-14.8%	1.01	161k
	FSDRP	20.7%	103.9	69.9	8.5%	1.02	173k
C = 4MB	FSSRP	4.2%	73.6	66	-18.4%	1.01	138k
	FSDRP	13.7%	73.6	53.6	4%	1.02	145k
C = 16MB	FSSRP	11.2%	169.5	141.2	-2.20%	1.01	153k
	FSDRP	13%	169.5	133.2	2%	1.03	198k

TABLE 6.7: Comparison analysis for different LLC capacity (C)

Param.	Policy	LI (%)	InterV Base	InterV FS	IntraV Red. (%)	Norm. EDP	Invalid-ation(k)
Ref. (A=16way)	FSSRP	14.7%	103.9	76.3	-14.8%	1.01	161k
	FSDRP	20.7%	103.9	69.9	8.5%	1.02	173k
A=8way	FSSRP	8.8%	149.8	130.3	-11.4%	1.00	132k
	FSDRP	16.5%	149.8	110.8	12.9%	1.01	151k
A=32way	FSSRP	5.4%	74.8	66.2	-4.9%	1.02	202k
	FSDRP	10.6%	74.8	58.3	1.6%	1.03	243k

TABLE 6.8: Comparison analysis for different LLC associativity (A)

in these cases, the inter-set write variation is marginally affected compared to the reference case.

6.5.4 Change in Capacity (C)

Table 6.7 lists the change in metrics values for different cache capacities. The size of the cache impacts the number of sets in the cache. Larger caches suffer from large inter-set variation compared to the smaller sized cache. This is because, in the case of larger caches with fixed associativity, the number of cache sets are large. With a large number of cache sets, there is a good possibility for the non-uniform write distribution across the cache set which in turn generates the inter-set write variation. However, in both the cases (for smaller and larger caches), our proposed schemes show considerable improvement in the coefficient of inter-set write variation and thus improves the lifetime.

6.5.5 Change in Associativity (A)

Table 6.8 shows the behavior of the proposed schemes with different associativity of the cache. Caches with lower associativity suffer from large inter-set write variation as compared to cache with larger associativity. This is because the

	FSSRP		FSDRP		
Cache Configuration	**m**	**r**	**m**	**r**	**I**
Small Size, Small Assoc.	↑	↓	↑	↓	= / ↑
Small Size, Large Assoc.	↓	↑	↓	↑	↓
Large Size, Small Assoc.	↑	↓	↑	↓	= / ↑
Large Size, Large Assoc.	↑	↑	↑	↑	↓

TABLE 6.9: Recommended values of m, r and I

cache with the same size and lower associativity have a large number of cache sets compared to the cache with higher associativity. As same as the previous case, the large number of cache sets introduces the uneven write distribution across the set. Although, in both cases, cache with lower associativity and higher associativity, our proposed schemes efficiently reduce the inter-set write variation and improves the lifetime, accordingly.

6.5.6 Recommended Values

Based on the above analyses, we recommend the values to be used for the m, r (in case of FSSRP) and I (in case of FSDRP) with the different configurations of caches. Table 6.9 lists these recommended values for the proposed schemes: FSSRP and FSDRP. Note that these recommended values are with respect to the reference values i.e. $m = 4$, $r = 4$ and $I = 5$ million cycles. The rationale behind the recommended values are presented below:

- **Interval (I):** To control the block residency, higher associativity needs small interval value. Whereas, the cache with lower associativity is not directly affected by I. Here, the value of I may be increased or kept the same according to the requirement.

- **Reserve ways (r):** In case of larger associativity, to handle the large write redirections from the NP, the value of r need to be increased. Whereas, the value of r has to be decreased for lower associativity.

- **Fellow Group Size (m):** Large-sized cache suffers from large write variation. For the removal of large write variation, the value of m needs to be increased.

However, in the case of a smaller cache, the value of m is decided according to the associativity and the window size.

6.6 Summary

Write variation inside the cache are affected by the applied replacement policy and the access patterns of the next generation applications running on many cores. In NVM based cache, this large write variation not only curtails the life but also diminishes the capacity of the cache over the period. This chapter presented efficient techniques to reduce the write variation across the cache sets called the inter-set write variation. Our first technique: FSSRP partitions the cache into groups of cache sets called fellow groups. Further, each group is logically divided into two parts: Normal and Reserve. To distribute the writes uniformly across the set, the normal part of the set can use the reserve part of the other sets in a fellow group. However, the major concern with the architecture of FSSRP is the increment in write concentration in the reserve part of the cache, which in turn increases the intra-set write variation, thus limiting the lifetime improvement. To overcome this shortcoming, our second technique: FSDRP, based on FSSRP, partitions the cache vertically into multiple equal-sized windows. During the execution, a different window is selected exclusively as a reserve part for a specified predefined interval. This helps to disperse the redirected writes over the cache. We examine the efficacy of the proposed approaches against the baseline STT-RAM and an existing technique: Swap Shift. Experimental evaluation with a full system simulator shows a significant reduction in the coefficient of inter-set write variation along with the lifetime improvement of 14.77% (FSSRP) and, 20.77% (FSDRP), respectively. Simultaneously, we also observe some reduction in intra-set write variation by FSDRP due to the dispersion of redirected writes. Thus, minimizing write variation inside the limited endurance non-volatile caches can make the system even more reliable and efficient.

भारतीय प्रौद्योगिकी संस्थान गुवाहाटी
Indian Institute of Technology Guwahati

Chapter 7

Improving the Performance of Non-Volatile and Hybrid Cache using Victim Caching

This chapter proposed methods to compensate for the performance gap caused due to the slow writes and the increase miss rate of NVM and the HCA. Towards this, the victim cache is integrated with both NVM and HCA by taking into account of various challenges with the employment of victim cache. The efficacy of the proposed methods is evaluated with baseline and the existing hybrid cache architecture.

7.1 Introduction

In the existing literature survey, many policies have been proposed to deal with the reduction of write energy and the weak write endurance of the caches made up of NVM technologies. But, at the same time, very less attention is paid to the performance loss (due to costly write operations) in these NVM caches. Section 7.1 shows the performance gap analysis in terms of percentage of CPI difference between the SRAM cache and the STT-RAM-based LLC for different types of workloads for

Work-load	PARSEC v2.1						SPEC CPU 2006			
Cache Config.	Cann	Ded	Fluid	Freq	Stream	X264	Mix1	Mix2	Mix3	Mix4
1MB	48.6%	7.1%	4.4%	3.63%	35.3%	16%	17.6%	12.2%	13.3%	10.2%
2MB	42.1%	5.4%	3.56%	2.37%	27.6%	11.9%	14.7%	11.8%	11.6%	9.53%
4MB	35.8%	2.2%	3.42%	1.75%	5.83%	11.1%	14.2%	10.3%	9.6%	9.3%
8MB	27%	1.8%	2.13%	1.66%	1.10%	8.6%	12.2%	8.82%	7.6%	7.85%

TABLE 7.1: Performance gap comparison between conventional SRAM and STT-RAM LLCs

different cache configurations (Details about the experimental setup are reported in section 7.5). From the table, we can conclude that the speedup reduction is the significant bottleneck towards NVM employment. Some of the previous techniques try to cope with the increased write latency by the use of Hybrid Cache Architecture (HCA) [27, 61]. In HCA, a small-sized SRAM partition is used to handle the write pressure of the cache, thereby saving energy and improving the performance. The only drawback with HCA is the increased miss rate on account of less residency of the block in write-intensive workloads and the limited capacity of the SRAM region. Table 2.2 shows this evidence in terms of percentage increase in miss rate by the existing technique: RWHCA [27, 28] against STT-RAM baseline for the different cache configuration against different workloads. The conclusion that can be derived from the table is that as the cache size becomes larger, the residency of the blocks increases, which in turn improves the miss rate. At the same time, due to increased miss rate, the smaller-sized caches (which are generally used in embedded systems) suffer from the performance.

To mitigate these performance gaps, this chapter proposes policies to improve the performance gap between the SRAM and Non-volatile cache/HCA by not changing the write behavior of caches but by associating an SRAM based fully associative victim cache [122] with the existing Non-volatile cache architecture/Hybrid Cache Architecture. As per our knowledge, none of the states of the art techniques exploit the use of victim cache towards reducing the performance gap in NVM caches and HCA. In particular, our proposed techniques differ in terms of the employment of victim cache with the type of memory technology used in the cache. However, some of the recent previous works [123, 124] tries to exploit victim cache with SRAM based LLCs to reduce the write operations in the NVM based main

memory system. With NVMs, our proposed victim cache architecture utilizes the write intensity of the blocks in victim cache to decide whether it is moved to the main cache or not upon a hit. Whereas, in HCA, the blocks evicted from main hybrid cache are stored in victim cache. During the regular cache access upon a miss in HCA, the VC is searched. If the required block is found in VC, it has to be moved to HCA. In this case, we propose a policy that intelligently places the blocks from the VC to an appropriate region of HCA depending on the type of request. We also propose to partition the VC dynamically into STT and SRAM region to balance the uneven block evictions from the different regions of the hybrid cache according to run-time load. In this chapter, we employed victim cache with the L2 as the hybrid LLC or pure LLC made of non-volatile memory technology. We use STT-RAM memory technology as non-volatile memory. However, our proposed technique can be easily extended to other non-volatile technologies such as PCRAM and ReRAM based cache.

The main contribution of this chapter is as follows:

- In NVM cache, upon hits to blocks in the victim cache, our policy selectively moves them to the main cache based on their write-intensity.
- In HCA, upon hit in the victim cache, we proposed a technique to intelligently place the required block in the appropriate region of HCA.
- In HCA, another technique is proposed that partitions the victim cache dynamically into two variable-sized regions to balance the uneven evictions from the different regions of HCA.
- Experimental evaluation on a full system simulator GEM-5 [118] shows significant performance improvement along with the savings in the execution time over the existing technique and the baselines.

Figure 7.1 presents the general overview of the proposed contributions: (a). Victim Cache with NVM cache (b). Victim Cache with Hybrid Cache in this chapter.

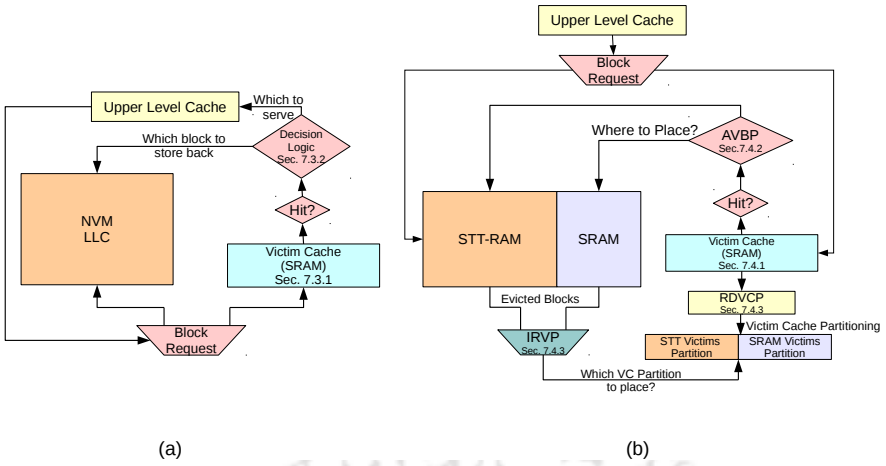


FIGURE 7.1: General overview of contributions: (a). Victim cache with NVM cache (b). Victim cache with hybrid cache in chapter 7

The rest of the chapter is organized as follows. Background and motivation are reported in section 7.2. Section 7.3 presents the technique to overcome the challenges imposed by integrating the victim cache with NVM cache. Section 7.4 Illustrates the methods to mitigate the obstacles generated by the integration of victim cache with HCA. Section 7.5 discussed the experimental evaluation. Results and analysis are reported in section 7.6. Comparative analysis with different configurations and parameters are presented in section 7.7. Finally, we conclude this chapter in section 7.8.

7.2 Background and Motivation

7.2.1 Victim Cache

The victim cache proposed by Jouppi [122] is used for improving the performance of SRAM based primary cache by retaining the victims evicted from the main cache. Usually, the victim cache is an SRAM based fully associative structure, and it is associated with any level of cache in the multi-level cache hierarchy. When the block is evicted from the main cache, it is retained into the victim cache by substituting the LRU block from the victim cache. When a block request (R) is received from the upper-level cache, the requested block is searched in both the

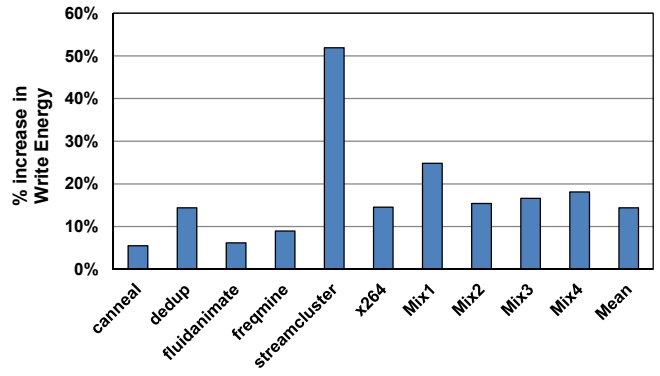


FIGURE 7.2: Percentage increase in write energy due to swap operation with victim cache

main cache and victim cache in parallel. If the block is found in the victim cache, the requested block is first placed into the main cache, and afterward, the request (R) is served. In case, if there is no invalid entry in the main cache, the LRU block from the cache set in the main cache is swapped with the requested block in the victim cache.

7.2.2 Motivation

Employment of the victim cache with the non-volatile cache requires excellent victim retention and block migration policy (from victim cache to the main cache). This is because migration incurs extra write cost and latency in the case of the non-volatile cache. Fig. 7.2 shows the increase in write energy due to swap/migration operation in between victim cache and non-volatile main cache for different workloads. These extra swaps or migration operations due to hits in the victim cache increase the dynamic energy by around 14.4% and adversely impact the expected performance gain offered by the use of victim cache. The conclusion that can be derived from the figure is that instead of migrating or swapping every block, if we selectively retain some of the blocks in the victim cache based on their write intensity, and entertain future requests to such blocks directly from the victim cache, we can further save the increase in write energy and improve the performance of the non-volatile cache.

Work-loads	PARSEC v2.1						SPEC CPU 2006				Mean
	Cann	Ded	Fluid	Freq	Stream	X264	Mix1	Mix2	Mix3	Mix4	
STT Placement	96.1%	99.8%	89.7%	60.3%	98.1%	53.6%	60.6%	89.3%	80.8%	97.7%	82.6%
SRAM Placement	3.9%	0.2%	10.3%	39.7%	1.9%	46.4%	39.2%	10.7%	19.2%	2.3%	17.4%

TABLE 7.2: Percentage times the block placed from victim cache to different region of hybrid cache

On the other hand, the employment of victim cache with hybrid cache requires good victim retention policy; and policy to place the blocks to the appropriate region of hybrid cache when they are moved back from the victim cache. This policy is needed because, upon a hit in the victim cache, there is a possible placement of the write-intensive victim blocks in the STT region of hybrid cache as this depends on the replacement policy of HCA, and that the LRU may be from STT region. This placement may incur extra writes to STT, which may degrade the performance as well as increase energy consumption. Such cases will overcome the benefits offered by the victim cache associated with the main hybrid cache. Section 7.2.2 shows the percentage times a block is placed in the STT region of main hybrid cache upon a hit in the victim cache (for different workloads). The conclusion that can be derived from the table is that, on average, 82.6% of the times block is placed in the STT region. This is due to the fact that there is a large possibility of LRU victim selection from the STT region as the cache set contains three fourth STT ways compared to one fourth SRAM ways. This motivates us to propose an effective and intelligent block placement policy to the appropriate regions of the hybrid cache upon a hit in the victim cache.

7.3 Integration of Victim Cache With Non-Volatile Cache

In this work, we propose to add SRAM based victim cache to NVM based main cache. The victim cache stores the victims as well as directly serves the request for write-intensive blocks instead of relocating them to NVM main cache. The decision to keep the block in victim cache is based upon the weight associated

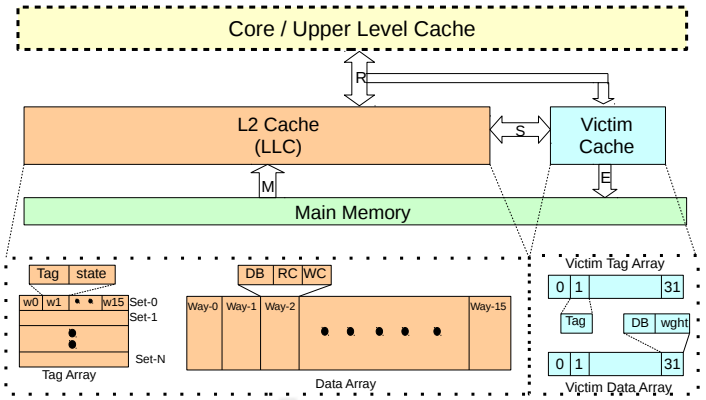


FIGURE 7.3: Schematic view of proposed victim cache architecture with non-volatile main cache

with each block. As the victim cache directly serves the request for individual blocks, the state of these blocks is maintained with directory entry of the main cache. These extra entries help in maintaining the coherence of the blocks. In our experimental evaluation, we update the LLC controller to handle the coherence of these additional blocks.

7.3.1 Architecture

Fig. 7.3 shows the representational view of victim cache architecture and its respective position in the memory hierarchy. In our architecture, the victim cache is associated with last level STT-RAM-based cache. We implemented victim cache as a fully associative structure that consists of both tag and data array as same as the normal cache. The data array of the victim cache consists of a victim data block and a 4-bit weight field. The weight is computed by giving three times more weight to write counts over read counts. This is because writes are more expensive in the non-volatile cache. The values of read and write counters are taken from the main cache at the time of eviction.

Algorithm 7 Write Latency aware Victim Caching (WLVC)

1: ULC: Upper Level Cache.
2: LLC: Last Level Cache.
3: I : Predefined interval.
4: Th: Threshold to decide write intensive block.
5: V_n: Total number of entries in victim cache.
6: List $< integer > wt$: List of weight entries associated with each block of victim cache. Size of list is V_n.
7: RC_{ij}: Read Counter with respect to set i and way j that records the number of read accesses from the ULC to the particular entry of LLC. $0 \leq i < S$, $0 \leq j < A$
8: WC_{ij}: Write Counter with respect to set i and way j that records the number of write accesses from the ULC to the particular entry of LLC. $0 \leq i < S$, $0 \leq j < A$
9: Run application for I cycles treating the LLC and victim cache as per their normal behavior.
10: **repeat**
11: **for** every interval I **do**
12: $Thr = Avg(wt[i]) + Bias$
13: **for** each request R from ULC to LLC **do**
14: **if** $R = ReadDirectHit$ **then**
15: The read operation is performed on block B in the LLC. Increment the corresponding RC_{ij} counter of the block.
16: **else if** $R = WriteDirectHit$ **then**
17: The write operation is performed on block B in the LLC. Increment the corresponding WC_{ij} counter of the block.
18: **else if** $R = ReadIndirectHit$ or $R = WriteIndirectHit$ **then**
19: Let k be the position of the block in the victim cache.
20: **if** $wt[k] \geq Th$ **then**
21: Serve request R from the victim cache.
22: $wt[k]++$
23: **else**
24: Swap the block with the LRU entry of main cache.
25: Serve request R from main cache.
26: Increment the corresponding counter associated with block in main cache.
27: **end if**
28: **else**
29: Forward request R to next level memory. ▷ Cache Miss
30: Evict LRU B' from main cache and load the incoming block.
31: Place B' in victim cache by evicting the WLRU.
32: Update weight of B' in victim cache.
33: **end if**
34: **end for**
35: **end for**
36: **until** the end of the execution

7.3.2 Operation

The operation of the proposed technique is described through the Algorithm 7. To discuss the algorithm in a better way, we consider a regular STT LLC of size M with S number of cache sets and associativity A and its associated fully associative victim cache of size N with V_n number of entries (line 5). The parameter I is used as a predefined interval (line 3). The weight associated with each victim entry in the victim cache is represented by the list wt of size V_n (line 6). Read and write counters associated with each block in the LLC is represented by two 2D arrays RC_{ij} and WC_{ij} (line 7 and 8).

For the initial I cycles of application execution, the LLC is treated as a usually

available cache, and its associated victim cache behaves as per their normal behavior (line 9). The read and writes counters are updated accordingly. Upon the block eviction from the main cache, the block move to the victim cache, and its weight value is calculated.

Our second aim is to retain the write-intensive block inside the victim cache. We defined the write-intensive blocks as those blocks weighting above average. Due to the limited capacity of the victim cache, we can retain a small percentage of write-intensive blocks. Hence, the threshold is defined as the summation of average weight entries of all victim entries and $Bias$ (line 12). The $Bias$ will be empirically calculated.

After I cycles are over, for every request R coming from upper-level cache, we perform a parallel search in the main cache and victim cache. If the block is found in the LLC, we call it a direct hit. Otherwise, if the block is found in the victim cache, we call it an indirect hit. Depending upon the type of requests and the result of the lookup operation, different operations are performed in the proposed architecture, which is described below:

- ***Read Direct Hit:*** The read operation is performed regularly on the block B in the LLC. Along with the read operation, the read counter RC_{ij} of block B is incremented (lines 14 and 15).

- ***Write Direct Hit:*** The write operation is performed normally on block B, and the corresponding write counter WC_{ij} is incremented (lines 16 and 17).

- ***Indirect Hit:*** In this case, the block B is found in the victim cache at location (say) k (line 18 and 19). If the weight entry of the block is equal or greater than Th (line 20), then the request R is served for the block B from its location in the victim cache (line 21). In this scenario, the corresponding weight entry of the B is incremented depending upon the type of request (+1 for read and +3 for write) (line 22).
On the other hand, if the weight entry of block B is less than Th (line 23), then the block in victim cache is swapped with the LRU position L of the LLC (line 24).

The weight of the new incoming block to the victim cache is updated accordingly. The request R is served by the LLC (lines 25 and 26).

- ***Cache Miss:*** In case, when the block is not found in either of the caches, i.e., main cache or victim cache, the Request R from ULC is forwarded to the next level of memory (main memory in our case) (line 28 and 29). The new incoming block is loaded in the main cache, and LRU B' is evicted (line 30). This B' is kept in the victim cache by removing Weighted Least Recently Used block (discussed in the next subsection) of victim cache (line 31). The weight entry of B' in the victim cache is updated using the read and write counters from the main cache (line 32). Note that with each miss in the victim cache, the weight value of all the entries is decremented by one. This keeps the most accessed and updated blocks in the victim cache.

7.3.3 Weighted Least Recently Used Replacement Policy (WLRU)

In place of conventional Least Recently Used replacement policy, we use WLRU for the block eviction in the victim cache. In the WLRU, the block is evicted according to the weight, as well as the timestamp. Weight has the priority to make a decision. The block whose weight is less than the other blocks is evicted first. In case, if two or more blocks have the same weights, then the decision is made according to the timestamp. This helps in retaining write-intensive blocks in the victim cache.

7.3.4 Working Example

The working example of WLVC for the indirect hit is presented in fig. 7.4. The figure shows the ULC, LLC, and victim cache having eight entries. Two cases are considered to demonstrate the method. In the first case, a write request (shown by dotted arrow-1) from the ULC to the way-1 of the victim cache is generally

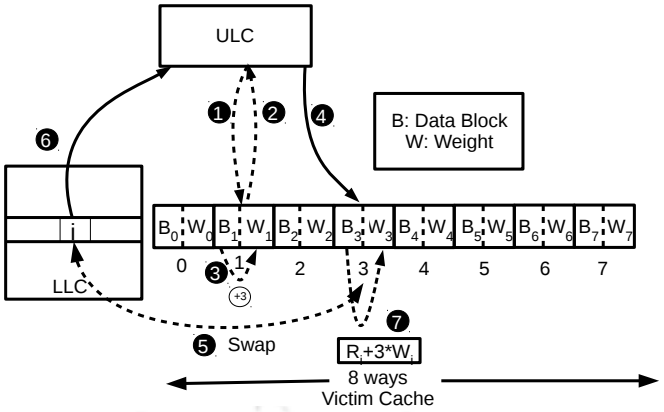


FIGURE 7.4: Working example of WLVC

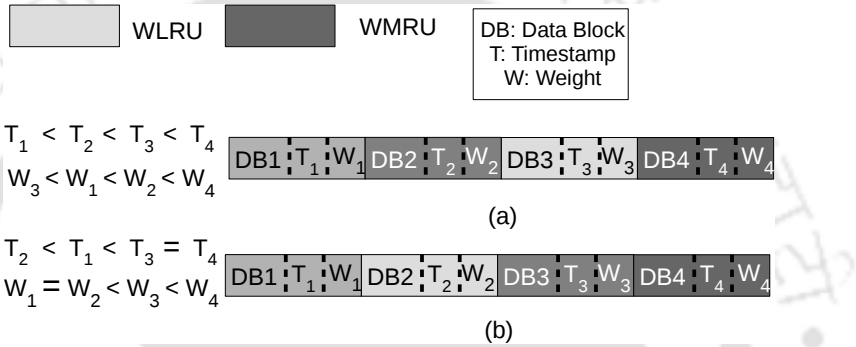


FIGURE 7.5: Working example of Weighted Least Recently Used replacement policy in a four entry victim cache (a) When all weights and time stamps are different (b) When some of the weights and time stamps are same

served from the original location as the weight entry of the block is greater than the *Th*. Once the request is served, the write-back acknowledgment is sent back to the ULC (arrow 2), and the weight entry of the block at way-1 is incremented by +3 (arrow 3). For the second case, the read request from the ULC to the way-3 of the victim cache (arrow 4) results in the swap operation between the LRU entry of the LLC and the requested block of victim cache (arrow 5). This is because the associated weight entry of way-3 is less than *Th*. Once the blocks are swapped, the request from the ULC is served from the LLC (arrow 6). Note that during the swap operation, the weight of the LRU entry of the LLC coming to the victim cache is updated with the help of read and write counters (arrow 7).

The working example of the Weighted LRU replacement (WLRU) policy is shown

in Fig. 7.5. The gray-scale of the blocks represents the stack position in the replacement queue with dark gray representing the maximum weight and the lightest color representing minimum weight. The explanation of the example in part (a) is straightforward. Among all the entries, the block DB3 is at the WLRU position in the stack, as its weight value is least among the other weight values. Hence victim is DB3. In the other case (part (b)), when some of the entries have the same weight, the WLRU position is decided based upon the timestamp associated with each entry of the cache. For example, DB2 becomes the WLRU because its timestamp is less than that of DB1.

7.4 Integration of Victim Cache with Hybrid Cache

In this work, our main aim is to improve the performance of HCA using a Victim Cache (VC), at the same time, maintain the principles of HCA that control the writes in the STT region. The supporting VC must obey this, and therefore we propose a policy that decides the partition in HCA when a block is moved back to HCA from VC called: Access-based Victim Block Placement (AVBP).
The theme of VC is to retain the most recent victims from the HCA. However, due to the uneven partition size of HCA, the evictions from HCA may be more from smaller partition: SRAM or will depend on application behavior. In case the SRAM partition of HCA performs more evictions over an interval, these evicted blocks, when moved to VC, will remove the blocks in VC belonging to the STT partition of HCA. To maintain a balanced mix of victims coming from individual partitions of HCA to the VC, we propose to partition the VC to hold blocks coming from each region. Depending on the increase or decrease in the number of evictions from SRAM, the size of partitions in VC is adjusted at run-time so that victims from STT get judicious space in VC. This policy is called Region-based Dynamic Victim Cache Partitioning (RDVCP).

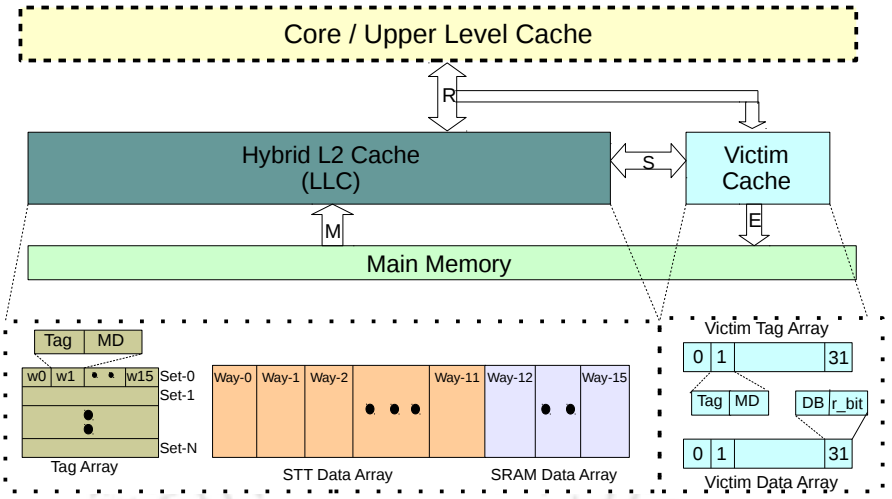


FIGURE 7.6: Schematic view of victim cache architecture associated with hybrid cache

7.4.1 Architecture

Figure 7.6 shows the schematic view of victim cache architecture associated with the main hybrid last level cache architecture. As shown in the figure, the hybrid cache is made up of a large number of STT ways and the small number of SRAM ways. The tag array (made up of SRAM) of main hybrid cache contains the tag and the metadata (MD) information: *state information* (to maintain the coherence of LLC block), *valid* and *dirty* bit for each block in the data array. The victim cache is a small SRAM based fully associative structure that consists of both tag and data array as a normal cache. The tag array of victim cache contains the tag and the MetaData (MD) information (a *valid* bit and a *dirty* bit) for each data block entry of victim cache. Note that when the block is evicted from the main hybrid cache and placed into the victim cache at the location, say T, the dirty bit at location T will be updated with the value of the dirty bit in the main cache. With each data entry in the data array of victim cache, a single bit: r_bit, is associated. The use of r_bit is to identify the region of the main hybrid cache from where the block was evicted. Note that, with each entry of victim cache, there is no need to maintain the coherence information as the blocks maintained in the victim cache is the evicted block from the main cache.

Algorithm 8 AVBP

```
1: ULC: Upper Level Cache.
2: LLC: Last Level Cache.
3: V_n: Total number of entries in victim cache.
4: repeat
5:     for every request R coming from ULC to LLC and miss in HCA along with hit in VC do
6:         Let the requested block B found in the victim cache at position m.
7:         if R == ReadHit then
8:             if B.dirty == 1 then
9:                 Swap B with LRU of SRAM region of the HCA.
10:            else
11:                Swap B with LRU of STT region of the HCA.
12:            end if
13:        else
14:            Swap B with LRU of SRAM region of the HCA.                ▷ WriteHit
15:        end if
16:    end for
17: until the end of the execution
```

7.4.2 Access based Victim Block Placement (AVBP)

This section elaborates on our proposed Access-based Victim Block Placement (AVBP) technique that places the block effectively and intelligently to the appropriate region of hybrid cache when found in the victim cache.

Operation: We elaborate on our proposed technique through the algorithm 8. The algorithm reports the case when the block is found in the victim cache, and it is to be placed into the appropriate region of the main hybrid last level cache. To explain the algorithm in an easy way, we consider a fully associative Victim Cache with V_n number of entries (line 3). For each request coming from Upper-Level Cache (ULC) to Last Level Cache (LLC), the tag lookup operation is performed in both the main hybrid cache and victim cache. Upon a hit in the main hybrid cache, the requested block is normally served as same as the normal cache. On the other hand, when the block is found in the victim cache, at position m (line 6) then according to the type of request, the block is swapped with appropriate regions of the main hybrid cache as described below:

- ***Read Hit:*** In this case, the dirty bit of the requested block B at position m in the victim cache is examined. If the requested block is found to be dirty, the block B is swapped (or moved if there is an invalid entry in the SRAM region of hybrid cache) with the LRU block of the SRAM region in the main hybrid cache. The reason behind putting the block in the SRAM partition is the multiple prospective future write requests for the requested block B, it being already dirty (lines 8 and

Work-loads	PARSEC v2.1						SPEC CPU 2006				Mean
	Cann	Ded	Fluid	Freq	Stream	X264	Mix1	Mix2	Mix3	Mix4	
$SR_E > ST_E$	99.4%	39.6%	19.6%	22.7%	4.31%	32.1%	19.3%	18.5%	13.2%	23.1%	29.8%

TABLE 7.3: Percentage times SRAM Eviction (SR_E) greater than the STT eviction (ST_E)

9). On the other hand, if the requested block is not dirty, the block is swapped with the LRU block in the STT region of the main hybrid cache (lines 10 and 11).

- ***Write Hit:*** In this scenario, the block is swapped with the LRU victim block of the SRAM region due to multiple prospective future write requests (line 13 to 15).

- ***HCA Miss and VC Miss:*** In case, if the block is not found in the main hybrid cache and the victim cache, then it is fetched from the main memory. As per the placement policy of the main cache, the fetched block is placed in the particular region, and the LRU block is evicted from that region. The LRU is kept in the victim cache. Note that, in this case, in order to make room for the evicted block of HCA, the LRU block of the victim cache is evicted.

Limitation: The limitation of AVBP is the larger number of evictions from the SRAM region on account of the proposed placement policy of HCA and the access behavior of running applications. In other words, the behavior of applications running on the multiple cores will not be constant, and it will change over the period. By experimental analysis, we have found that within an interval of two million cycles, sometimes the eviction from the SRAM region is greater. Section 7.4.2 presents this evidence where the number of times the SRAM region eviction is greater than the STT region eviction throughout two million intervals for the whole execution. As reported in the table, on an average 29.8% of times, the evictions from the SRAM region are higher than the STT region, and these evicted SRAM blocks will replace the STT block of VC. This can cause the victims evicted from the STT-RAM will not stay back longer in the victim cache and get prematurely evicted from thereself. This motivates us to propose a dynamic region-based victim cache partitioning technique to control this uneven SRAM eviction behavior and keep judicious space in VC for STT blocks.

7.4.3 Region based Dynamic Victim Cache Partitioning (RDVCP)

Main Idea: The key idea of RDVCP is to intelligently partition the victim cache dynamically into two regions: one for SRAM victims and one for STT victims. The victim evicted from the main hybrid cache is placed appropriately in one of these victim regions. The sizes of the SRAM victim region and STT victim region are adjusted at run-time depending on the application pattern. Note that the dynamic decision for altering the sizes is taken after every predefined interval I.

Operation: We explain the operation of RDVCP through the algorithm 9. The algorithm describes the interval wise Region-based Dynamic Victim Cache Partitioning and the placement of the evicted block from the main hybrid cache to the different regions of victim cache. Note that while placing the victims, the algorithm is maintaining the allocated partition sizes for the current interval. Similar to algorithm 8, the functionality of parameter V_n is the same (line 2 of algorithm 9). The tunable parameter I is used as a predefined interval for making the decision of dynamic victim cache partitioning (line 3). The threshold used for altering the allocated partition sizes of the victim cache is represented by the parameter $Bias$[1] (line 4) at the end of each interval. The count of the total number of evictions from the SRAM region of the main hybrid cache in the current interval and in the previous interval is represented by the variables $Curr\_SRAM\_Evict$ and $Prev\_SRAM\_Evict$ respectively (line 5 and 6). The variables vic_STT_ways and vic_SRAM_ways are used to maintain the count of the number of victim ways allocated to the STT-RAM victim region and SRAM victim region respectively for the current interval (line 7 and 8). Initially, at the beginning of execution, half of the ways of the victim cache is allocated to each region (line 9). For a block in victim cache, to identify from which region in the main hybrid cache it came

[1]We have found that the value of $Bias$ by empirical analysis by conducting an extensive profiling for the different set of values as reported in section 7.7.3

Algorithm 9 RDVCP

HCA: Hybrid Cache Architecture
V_n: Total number of entries in victim cache.
I: Predefined Interval.
$Bias$: Threshold to change the partition size of VC.
$Curr\_SRAM\_Evict$: Eviction counter that records the number of eviction from SRAM in the current interval.
$Prev\_SRAM\_Evict$: Eviction counter that maintains the eviction count from the previous interval.
vic_STT_ways: Number of ways allocated in VC for STT region.
vic_SRAM_ways: Number of ways allocated in the VC for SRAM region.
$vic\_STT\_ways = V_n/2$; $vic\_SRAM\_ways = V_n/2$
$max\_SRAM\_vic = 3V_n/4$; $min\_SRAM\_vic = V_n/2$
Run application for I cycles treating the whole cache as a normal cache and the victims evicted from each region of HCA are stored in the respective partition of VC.
repeat
  for at the end of every interval I **do**
    Let $\delta = Curr\_SRAM\_Evict - Prev\_SRAM\_Evict$
    Let $\delta' = Prev\_SRAM\_Evict - Curr\_SRAM\_Evict$
    $x = vic\_SRAM\_ways + V_n/4$; $x' = vic\_SRAM\_ways - V_n/4$
    $y = vic\_SRAM\_ways + V_n/8$; $y' = vic\_SRAM\_ways - V_n/8$
    if $\delta \geq 2 * Bias$ **then**
      if $x \leq max\_SRAM\_vic$ **then**
        $vic\_SRAM\_ways = x$
      else if $y \leq max\_SRAM\_vic$ **then**
        $vic\_SRAM\_ways = y$
      end if
    else if $Bias \leq \delta < 2 * Bias$ **then**
      if $y \leq max\_SRAM\_vic$ **then**
        $vic\_SRAM\_ways = y$
      end if
    else if $\delta' \geq 2 * Bias$ **then**
      if $x' \geq min\_SRAM\_vic$ **then**
        $vic\_SRAM\_ways = x'$
      else if $y' \geq min\_SRAM\_vic$ **then**
        $vic\_SRAM\_ways = y'$
      end if
    else if $Bias \leq \delta' < 2 * Bias$ **then**
      if $y' \geq min\_SRAM\_vic$ **then**
        $vic\_SRAM\_ways = y'$
      end if
    end if
    $vic\_STT\_ways = V_n - vic\_SRAM\_ways$ size.
    $IRVP(vic\_STT\_ways, vic\_SRAM\_ways)$
  end for
until the end of the execution

Interval wise Region based Victim Placement

function IRVP(new_STT_ways, new_SRAM_ways)
  $exist\_STT\_ways$: Existing allocation of STT victim counts.
  $exist\_SRAM\_ways$: Existing allocation of SRAM victim counts.
  for every eviction of block B in HCA **do**
    Let the block B evicted from the region P of HCA.
    if $new\_P\_ways \leq exist\_P\_ways$ **then**
      Place B to its respective region P of VC.
    else
      Place B to the other region P' of VC.
      $exist\_P\_ways - -$; $exist\_P'\_ways + +$
      Update r_bit for B.
    end if
  end for
end function

from, we use a bit called *r_bit*. If the *r_bit* is set, the block has come from the STT region; else from the SRAM region.

We propose the maximum and minimum (line 10) size allowed for each victim region as follows:

- $max\_SRAM\_vic = 3V_n/4$, $min\_SRAM\_vic = V_n/2$
- $max\_STT\_vic = V_n/2$, $min\_STT\_vic = V_n/4$

For the initial I cycles of application execution, the victim cache behaves normally with each block evicted from the main hybrid cache stored in the respective region of the victim cache (line 11). Once the application crosses the I cycles, i.e., **at the end of interval**, different operations are performed according to the uneven evictions from the SRAM region of the main hybrid cache:

- ***The SRAM evictions in the current interval are greater than the Previous interval:*** In this case, depending on the difference (δ) in the values of current interval SRAM eviction and the previous interval SRAM eviction, an appropriate increase in the SRAM victim region is performed. Also, note that increment in the SRAM partition results in the corresponding decrease in the STT partition of VC (line 39). Specifically, if $\delta \geq 2 * Bias$, we increase SRAM part by at most $V_n/4$ (lines 18 to 20). If the increase in the value of victim regions violates a maximum constraint, then an increase of $V_n/8$ is performed (lines 21 to 23). Further, in case, if the partition size is already at the maximum limit, no change in size is performed.
On the other hand, if the difference $Bias \leq \delta < 2 * Bias$ an increase by $V_n/8$ is performed to the size of the SRAM victim region (lines 24 to 26).

- ***The SRAM evictions in the current interval are less than the previous interval:*** Using the same logic as in the above case, a decrease of SRAM victim region size is performed keeping the minimum size constraint (lines 28 to 38).

Note that the above operations of the dynamic victim cache partitioning are performed at the end of each interval until the end of application execution (line 42). This decides the new sizes of the partition for the next interval. In the meanwhile, between the intervals, the evicted block from the main hybrid cache will be placed to the appropriate regions of the victim cache according to the new sizes of the victim regions (line 40). If the size of victim region changes, then there may be a case that SRAM blocks are in the STT region of victim cache and vice-versa. To stabilize the region with the correct block, we proposed an **Interval wise Region-based Victim Placement (IRVP)** algorithm.

In the algorithm, the existing status count for each region in the victim cache is represented by the variables $exist\_STT\_ways$ and $exist\_SRAM\_ways$ respectively (lines 44 and 45). Let the block B be evicted from HCA from region P (the region other than P of HCA is represented by P') (line 46 and 47), to place B in VC; two cases are described below:

- ***The new size of the P is less than or equal to the existing count:*** In this case, the block B is placed in the region P of the victim cache by victimizing the LRU block from P. Note that, in this case, for identifying the region of the block for victim selection, the r_bit is used (lines 48 and 49).

- ***The new size of the P is higher than the existing count:*** Here, the block B is placed in the region P' of the victim cache by evicting the LRU block (at the location, say T) from P'. Once the block B is placed, the respective r_bit at location T is accordingly updated (line 50 to 54).

7.4.4 Working Example

Figure 7.7 shows the working example of Interval wise Region-based Victim Placement. In the example, an eight entry fully associative victim cache is considered that consist of the victim data block (represented by SR for SRAM region block and ST for STT region block) and it's associated r_bit. Two cases are considered

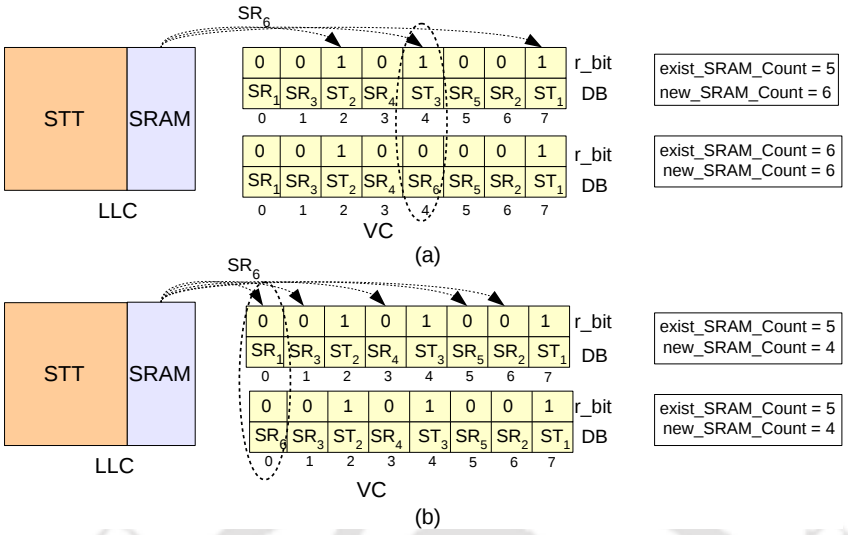


FIGURE 7.7: Working example of Interval wise Region based Victim Placement. (a) $new > existing$ (b) $new \leq existing$

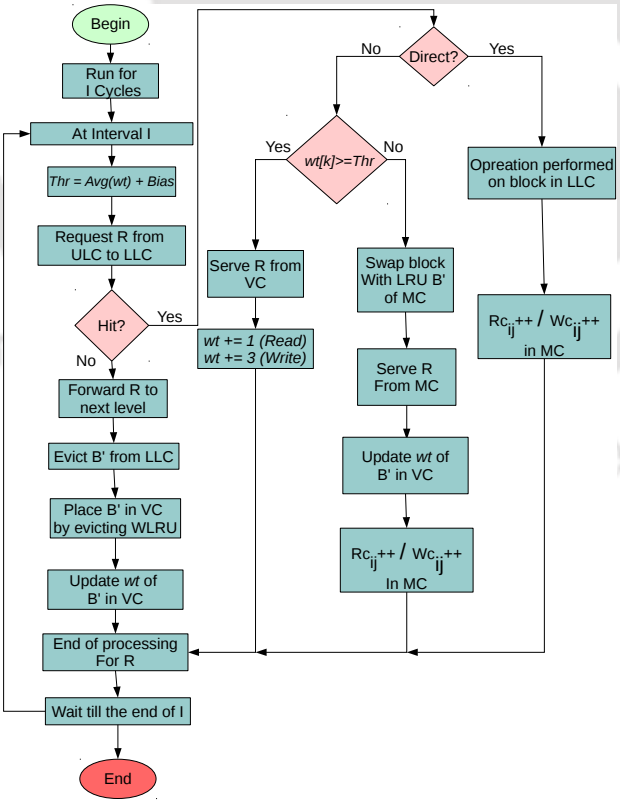


FIGURE 7.8: WLVC working flow chart

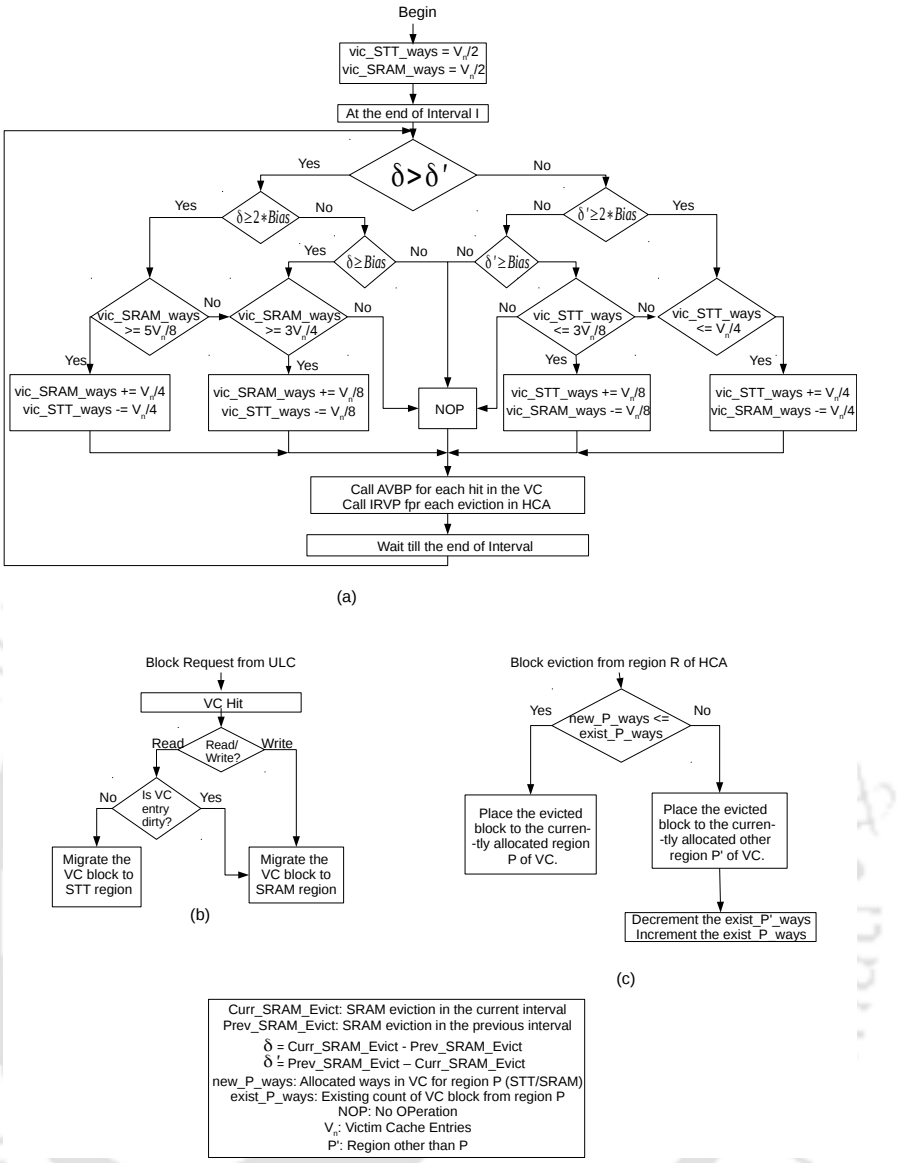


FIGURE 7.9: (a). Working flowchart summarizing the proposed schemes: RD-VCP and AVBP (b) Flowchart representing AVBP (c) Flowchart representing the IRVP

to demonstrate the method. In the first case (a), the existing count of the SRAM victim region blocks in the victim cache is lesser than the new SRAM victim region count in the current interval. In such conditions, when the block SR_6 is evicted from the SRAM region of the hybrid LLC, then the evicted block is placed in the STT victim region of the victim cache at LRU position 4 and *r_bit* is made to zero. On the other hand, in the second case (b), the existing status count of the SRAM victim region block in the victim cache is greater than the new SRAM victim region count of the interval. Here, when the block SR_6 is evicted from the

Components	Parameters
Processor	2Ghz, Quad Core, X86
L1 Cache	Private, 32 KB SRAM Split I/D caches, 4-way set associative cache, 64B block, 1-cycle latency, LRU, write-back policy
L2 Cache	Shared, 16-way STT/Hybrid (12-way STT-RAM and 4-way SRAM) set associative cache , 64B block, LRU, write-back policy
Victim Cache	SRAM, fully-associative, 64B block, WLRU/LRU policy
Main Memory	2GB, 160 cycle Latency
Protocol	MESI CMP Directory

TABLE 7.4: System parameters

LLC, it will be placed in the respective SRAM region by evicting its LRU blocks say at location 0, and *r_bit* remains the same.

Figure 7.8 summarizes the working flow chart of the proposed approach: WLVC. Figure 7.9 shows the summarized working flow diagram of proposed approaches: AVBP and, RDVCP during application execution in the CMP system. The fig. 7.9 (a) shows the approach used during the RDVCP. While, fig. 7.9 (b) presents the working of AVBP and fig. 7.9 (c) shows the interval wise region-based victim placement.

7.5 Experimental Methodology

7.5.1 Simulator Setup

We implemented our proposed approaches on a full system simulator GEM-5 [118]. Table 7.4 shows the system parameters used in our simulations. The experiments are conducted on the different configurations of Hybrid LLC and STT-RAM LLC (L2 cache) and with different victim cache sizes. CACTI [19] and NVSIM [21] simulator obtains the timing, energy, and area parameters of these configurations at the 32nm technology node. Table 7.5 reports these values. Note that we have also considered the energy consumption (including static and dynamic) along with the latency overhead and circuit cost of victim cache in our experiments. The area overheads are reported and modeled using NVSIM.

LLC/ VC Size	LLC Type	Static Power (mW)	Read Energy (nJ)	Write Energy (nJ)	Read Latency (ns)	Write Latency (ns)
LLC Size = 1MB	SRAM	138.7	0.116	0.116	1.874	1.874
	STT	59.44	0.188	2.117	2.117	11.34
	Hybrid	79.25	0.188/ 0.116	2.117/ 0.116	2.117/ 1.874	11.34/ 1.874
LLC Size = 2MB	SRAM	282.2	0.221	0.221	2.00	2.00
	STT	92.68	0.285	2.147	2.336	11.54
	Hybrid	140.1	0.285/ 0.221	2.147/ 0.221	2.336/ 2.00	2.336/ 2.00
LLC Size = 4MB	SRAM	554.3	0.330	0.330	2.180	2.180
	STT	229.6	0.346	2.270	2.577	11.94
	Hybrid	310.8	0.346/ 0.330	2.270/ 0.330	2.577/ 2.180	11.94/ 2.180
LLC Size = 8MB	SRAM	1094.5	0.432	0.432	2.043	2.043
	STT	342.0	0.693	2.355	3.177	12.41
	Hybrid	530.1	0.693/ 0.432	2.355/ 0.432	3.17/ 2.043	12.41/ 2.043
Victim Cache						
VC Size = 16 Entries	SRAM	2.93	0.007	0.007	0.240	0.240
VC Size = 32 Entries		5.48	0.009	0.009	0.307	0.307
VC Size = 64 Entries		10.44	0.014	0.014	0.416	0.416
Iso Area Analysis						
LLC Size = 6MB	STT	285.8	0.520	2.31	2.87	12.17

TABLE 7.5: Timing and energy parameters for different LLC and VC configurations

We compared our proposed approach against different baselines, existing approach, and with the different variety of proposed approaches (in case of hybrid cache) as mentioned below:

- **Base pure STT and Base pure SRAM:** The baseline architecture with no data placement policy and uses Least Recently Used (LRU) as a replacement policy.

- **STT Main Cache with Victim Cache (VC):** The baseline STT based main cache architecture with integrated victim cache having no selective caching scheme and uses LRU as a replacement policy.

- **STT Main Cache with Write Latency aware Victim Cache (WLVC):** STT-RAM-based Non-volatile Cache with the support of caching the write-intensive block in the victim cache. The victim cache uses WLRU as a replacement policy.

- **Base HCA:** The baseline hybrid cache architecture with integrated victim cache having no data placement policy and uses LRU as a replacement technique.

- **RWHCA [27]** (denoted by **R**): An existing data placement approach that places data according to the type of access to the different regions of HCA. Here,

2-bit counters are used to capture the accesses of the block. When there is a disproportion in the accesses in any of the regions, it leads to the migration of blocks.

• **HCA with Victim Cache and Placement policy (HCAVP)** (denoted by **O**): The baseline HCA that places the data to different regions upon LLC miss as same as RWHCA. In this variety, the only difference with RWHCA is that there are no counters associated with the blocks, and there is no migration process.

• **HCAVP with AVBP:** (denoted by **P**): Hybrid cache architecture with the full support of placement policy that includes the placement of block from victim cache upon hit to different regions of HCA.

• **HCAVP with RDVCP:** (denoted by **Q**): Hybrid Cache Architecture that includes the initial block placement from main memory upon LLC miss and the proposed region-based dynamic victim cache partitioning approach.

• **HCAVP with AVBP and RDVCP** (denoted by **S**): Hybrid cache architecture integrated with all the proposed approaches.

In our simulations, we have considered the extra time taken for the accesses and searching in the victim cache in the experiments. In particular, five cycles are taken during the block swapping between the main hybrid/STT main cache and the victim cache, and one cycle is taken for the searching of the block in the victim cache. This is the extra cycle, as the search has already begun in parallel with the main cache. The writing and the placement of the evicted LRU blocks from the main STT/HCA to the victim cache will not fall into the critical path as the victim cache is an independent structure and is not affected when a new request comes to the main cache. We consider the 42-bit swap buffers in our experiments for the tag swap operations.

In WLVC, we have set the size of the counters (RC_{ij} and WC_{ij}) associated with each block of main cache set to 2-bit[2]. Note that these counters (RC_{ij} and WC_{ij})

[2]Before choosing the size, we perform extensive analysis on the set of PARSEC and SPEC benchmark suites and found out that in most of the cases both the read and write hits can be accommodated in a 2-bit counter.

Benchmark suite	Benchmarks
PARSEC v2.1	Canneal (Cann), Dedup (Ded), Fluidanimate (Fluid), Freqmine (Freq), Streamcluster (Stream) X264
SPEC CPU2006	**Mix1:** milc, hmmer, bzip2, soplex (High WBKI) **Mix2:** dealii, sjeng, h264ref, tonto (Random mix) **Mix3:** gobmk, tonto, sjeng, namd (Mid WBKI) **Mix4:** calculix, astar, dealII, h264ref (Low WBKI)

TABLE 7.6: Benchmarks used for evaluation

are the saturating counter and made up of SRAM memory technology. On the other hand, the size of the weight field (wt) and timestamp associated with each block of the victim cache is set to 4 and 5-bits. The value of weight counter (wt) is updated with each read and write in the victim cache as well on a miss. Note that the extra time required for computing and updating the counters (including RC_{ij}, WC_{ij} and wt) is not in the critical path (as it is done in parallel with writing and reading of the block).

Whereas in RDVCP, to measure the evictions from the SRAM region, we added two 12-bit counters, and to maintain the existing count and new count of the victim region, four 5-bit counters are used.

7.5.2 Workloads

The proposed techniques are examined by using both multi-threaded: PARSEC [6] and multi-programmed: SPEC CPU 2006 [7] benchmark suites. Six benchmarks with *medium* input set are taken from PARSEC benchmarks and twelve benchmark with *ref* input set are used from SPEC. Section 7.5.1 lists the mixes of applications. The mixes (mix1, 3, and 4) composed of the multi-programmed benchmarks are based upon the Write-Back per Kilo Instruction (WBKI). Whereas, the mix2 shows the random composition of multi-programmed benchmarks.

7.6 Results and Analysis

Out of the different configurations of LLC and VC sizes, we select 4MB 16-way set associative last level hybrid/STT main cache and 32-way (V_n) fully associative

Policy	Region	O	P	Row
RWHCA	STT	35.8%	39.6%	1
	SRAM	-34.6%	-37.1%	2
	Total	5.52%	5.51%	3
Base HCA	STT	56.5%	59.1%	4
	SRAM	-171.5%	-176.5%	5
	Total	-0.84%	-0.85%	6
Base STT	STT	66.3%	68.3%	7
	Total	-1.91%	-1.92%	8
Base SRAM	SRAM	38.4%	37.3%	9
	Total	-0.80%	-0.82%	10

TABLE 7.7: Percentage savings in write accesses (higher is better)

victim cache. Similarly, the 4MB 16-way (partitioned into two regions: 12-way STT and 4-way SRAM) associative hybrid cache architecture is considered for RWHCA. In the simulations of WLVC, we set the values[3] of I to 1M cycles and $Bias$ to +1. Whereas, in the simulations for RDVCP, we choose the value of $Bias$ to 500 and I to 1M cycles. The reason behind selecting these values and cache configurations are explained in the later section, where we present brief results with the different cache configurations and parameters values. We show the effects on the following metrics: Write Savings (for only HCA), CPI Improvement, Execution time, Miss Rate, and Energy Overhead.

7.6.1 Write Accesses

Figure 7.10 presents the normalized write accesses against RWHCA. Section 7.6 presents the percentage savings in write access by the techniques: O and P over RWHCA and the base STT, SRAM, and HCA. Note that the negative value in the table (row 2, 5, 8, and 10) imply the increase in writes. It shows that in the proposed techniques, a larger number of writes are performed in the SRAM region of HCA (row 2 and 5).

With respect to RWHCA, the saving in write accesses by policy P (39.6%) is basically due to lesser writebacks by AVBP (row 1). However, a large number of writebacks are redirected to the SRAM region (-37.1%) (row 2). Note that the

[3]We conducted extensive profiling for the different sets of values for I. And, accordingly, we selected the most stable values.

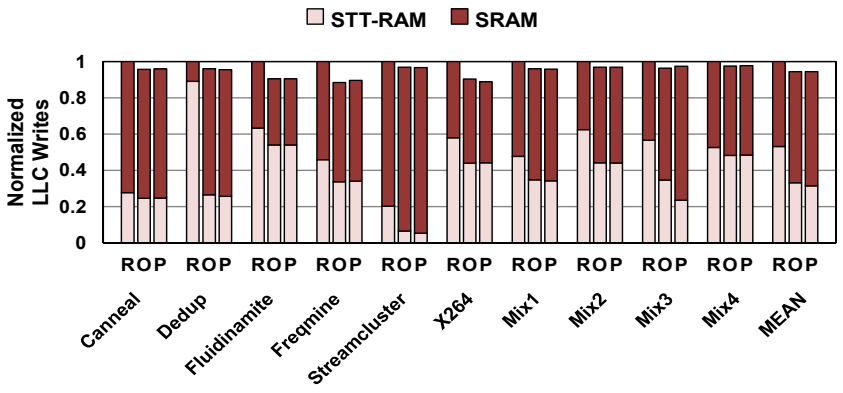


FIGURE 7.10: Normalized LLC writes of O and P against RWHCA (R) (lower in STT is better)

improvement in the total write accesses (row-3) is basically due to the improvement in the STT region as we have a 3:1 ratio in the hybrid cache of STT versus SRAM.

Compared to Base HCA, the saving in the write accesses for STT region (59.1%) (row 4) and the increase in the write for the SRAM region (-176.5%) (row 5) are due to the appropriate placement of the different types of blocks in the different regions of HCA.

Also, over the baselines, the region-wise significant gain is observed. But at the same time, due to proper placement by the AVBP, the proposed technique maintain the same number of writes with the marginal increase (row 7, 8, 9, and 10).

Note that the write accesses results are not discussed for the policy Q and S as they maintain results similar to O and P. Policies Q and S are policies for VC optimization, and they do not affect STT writes.

Effect on endurance: With a lesser number of writebacks, our proposed technique improves the endurance (measured in terms of the number of writes in the STT region, write traffic reduction) of the non-volatile region of the cache by savings the writebacks. In particular, the saving in the writeback traffic for P in the STT region are 42.7% respectively over RWHCA.

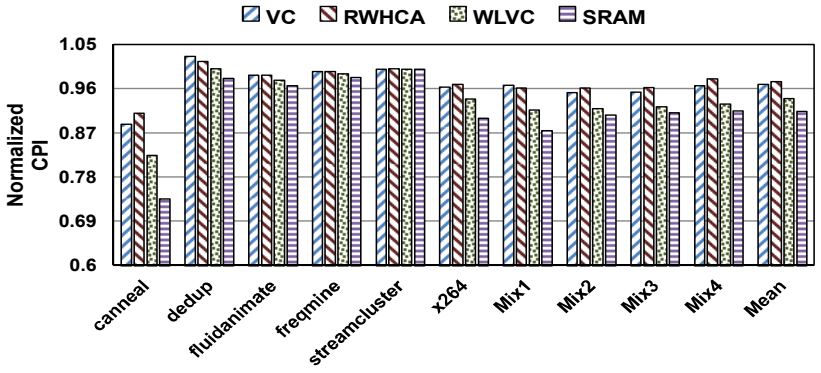


FIGURE 7.11: Normalized CPI with respect to Base STT-RAM (lower is better)

Policy	O	P	Q	S	Row
RWHCA	1.35%	1.4%	1.70%	2.32%	1
Base HCA	2.08%	2.20%	2.42%	3.03%	2
Base STT	3.52%	3.60%	3.85%	4.43%	3

TABLE 7.8: Percentage improvement in CPI (higher is better)

7.6.2 CPI Improvement

7.6.2.1 Effect on NVM based Main Cache

Fig. 7.11 shows the normalized performance of the different techniques concerning Base STT-RAM. Our proposed method: WLVC improves the performance by 5.88% over Base STT, 3.45% against RWHCA, and 2.95% over VC. The performance gain against Base STT is due to the retention of the victims in the victim cache, which in turn saves costly memory access time upon hit. On the other hand, the further speedup is due to the serving of request for write-intensive blocks directly from the SRAM based victim cache. Compared to RWHCA, the improvement in CPI by WLVC is basically due to increased evictions by the existing technique due to the partitioning of cache and the large data placement in the limited sized SRAM region. Thus, replacing the SRAM main cache with STT-RAM degrades performance by 9.3%. Using our proposed policy, this degradation is brought down to 2.87%.

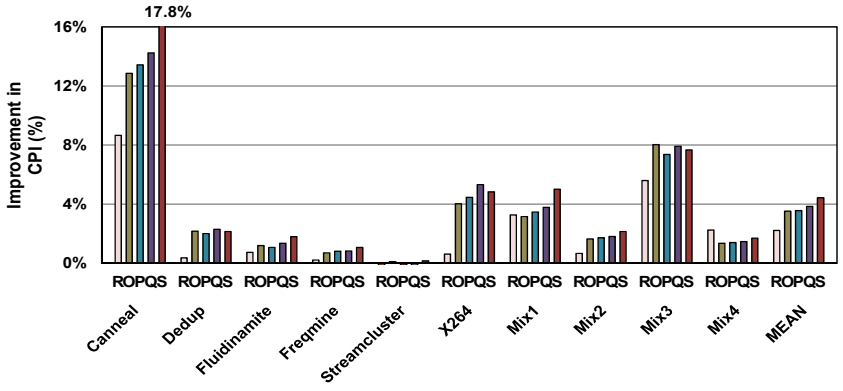


FIGURE 7.12: Percentage improvement in CPI of R, O, P, Q and S against Base STT (higher is better)

7.6.2.2 Effect on HCA based Main Cache

Figure 7.12 shows the percentage improvement in CPI concerning base STT. Section 7.6.2.1 reports the percentage improvement values in performance against different techniques: RWHCA, Base STT, and Base HCA. Over RWHCA, the improvement (2.32%) is due to applied victim cache and the proposed policies and the fewer number of writes operations (row-1). Compared to Base-HCA and Base-STT (row-2 and 3), the improvements (3.03% and 4.43%) are due to the placement policy, the victim cache with its proposed techniques, and large savings in the write overhead in case of Base STT.

7.6.3 Execution Time Improvement

7.6.3.1 Effect on NVM based Main Cache

The normalized execution time concerning Base STT-RAM is shown in Fig. 7.13. WLVC gets execution time gain of 5.43% with respect to Base STT, 3.38% concerning RWHCA, and 2.90% against VC. These gains in execution time are because we retain both victims and write-intensive blocks in victim cache and serve requests directly from there. This reduces the overall memory access time due to less off-chip access and saves on the swap overhead. In particular, we save 12.63% on swaps. Thus, replacing SRAM based main cache with STT-RAM-based main

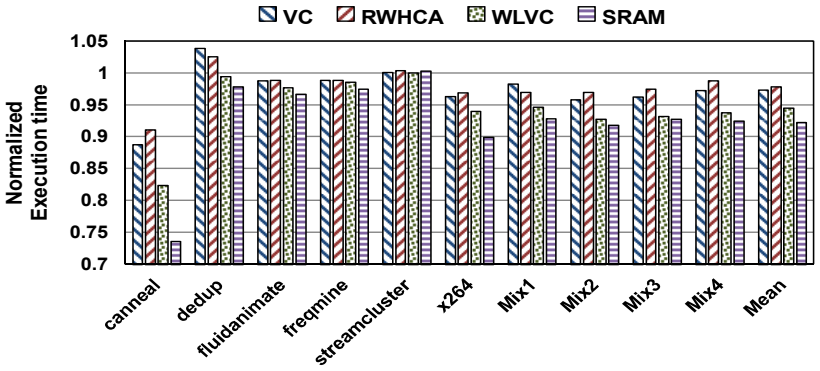


FIGURE 7.13: Normalized execution time with respect to Base STT (lower is better)

Policy	O	P	Q	S	Row
RWHCA	1.42%	1.44%	1.70%	2.26%	1
Base HCA	2.23%	2.25%	2.51%	3.05%	2
Base STT	3.39%	3.41%	3.65%	4.18%	3

TABLE 7.9: Percentage improvement in execution time (higher is better)

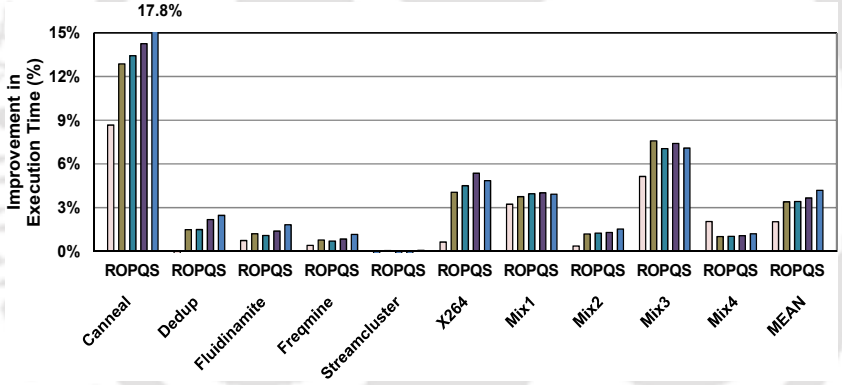


FIGURE 7.14: Percentage improvement in execution time of R, O, P, Q and S against Base STT (higher is better)

cache degrades the execution time by 8.3%. By using WLVC, this degradation is lower down to 2.44%.

7.6.3.2 Effect on HCA based Main Cache

The percentage improvement in execution time against Base STT is shown in figure 7.14. Section 7.6.3.1 presents these improvement values against the existing technique: RWHCA and the baselines HCA and STT. The improvements in the execution time (2.26% to 4.18%) (row 1-3) are due to the improvement in the

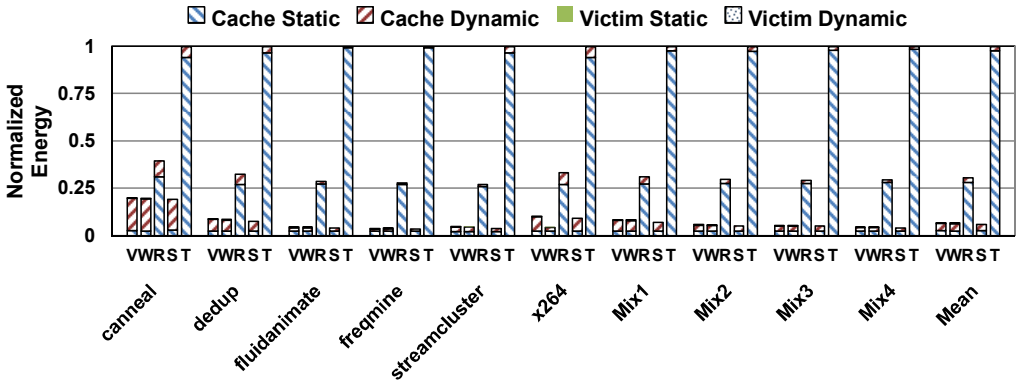


FIGURE 7.15: Normalized energy of VC (V), WLVC (W), RWHCA (R) and STT (S) with respect to Base SRAM (T) (lower is better)

CPI and the reduction in the miss rate that in turn decreases the main memory accesses.

Thus, by the use of Base pure STT as LLC, there is a degradation in performance and execution time due to the costly write operations. The use of RWHCA can overcome this overhead. However, with the proper placement policy in the HCA and by the use of VC with the proposed policies, the performance can be improved further.

7.6.4 Energy Consumption

7.6.4.1 Effect on NVM based Main Cache

Due to extra swap or migration operation and the accesses in the associated victim cache structure, our proposed technique: WLVC consumes slightly more energy compared to Base STT-RAM, as shown in Fig. 7.15. The respective percentage overhead of total energy and dynamic energy of WLVC against Base STT is 8% and 9.73%. But at the same time, due to performance gain, the savings in static energy against Base STT is 5.43%. Concerning Base SRAM, the savings in energy are 93.5% due to near-zero leakage power of STT-RAM. Concerning RWHCA, WLVC gets an overall energy saving by 78.85%. These significant energy savings are due to the large leakage energy of the SRAM part incorporated with

Policy	Region	O	P	Row
RWHCA	STT	26.6%	28.2%	1
	SRAM	-100.4%	-104.7%	2
	Total	13.77%	14.6%	3
Base HCA	STT	33.9%	35.2%	4
	SRAM	-37%	-40%	5
	Total	23.2%	24%	6
Base STT/ SRAM	STT	33.2%	33.8%	7
	SRAM	41.8%	41.8%	8
	Total	40.7%	40.8%	9

TABLE 7.10: Percentage improvement in the energy consumption (higher is better)

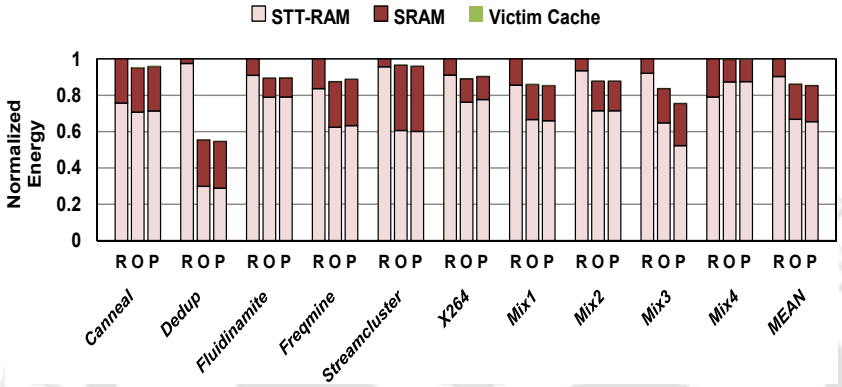


FIGURE 7.16: Normalized LLC energy consumption of O and P against RWHCA (R) (lower is better)

the RWHCA. However, due to larger write entertainment by the SRAM partition in the RWHCA, WLVC consumes more dynamic energy than RWHCA by 45%. Compared to VC, the percentage savings in static and dynamic energy due to lesser execution time and smaller swaps by the proposed technique is 2.9% and 5.98%, respectively.

7.6.4.2 Effect on HCA based Main Cache

The figure 7.16 shows the energy by the proposed techniques normalized with respect to Base STT. Table 7.10 reports these improvement values. Note that negative values (row 2 and 5) in the table implies the increase of energy. Over RWHCA, the improvement of 14.6% in energy consumption is basically due to lesser write operations (due to AVBP) (as evident from section 7.6) and no migrations (that incurs extra energy) in the proposed technique. However, the increase

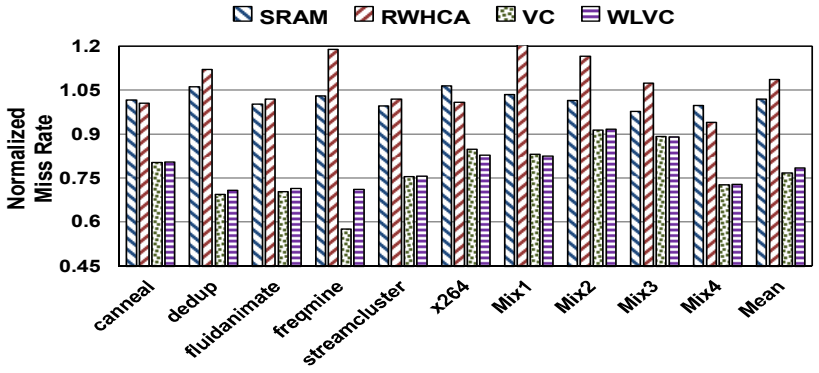


FIGURE 7.17: Normalized miss rate with respect to Base SRAM (lower is better)

in energy consumption in the SRAM region is due to large write accesses (as shown in section 7.6). Compared to Base HCA, the energy improvements (35.2% in row-4 and 24% in row-6) are due to the appropriate placement of blocks in the different regions. The same reason is applied to the energy increase in SRAM (row-5). Note that the energy improvement values given at row 7 (33.8%) are the dynamic energy improvement against the baseline STT. On the other hand, the values are given at row 8 (42.3%), and 9 (41.3%) are against the SRAM for static energy and total energy consumption. Note that we have also considered the energy consumption due to victim cache (both static and dynamic energy) in our calculations. Note that the energy consumption results are not discussed for the policy Q and S as they maintain the same number of write accesses with O and P in the main hybrid cache.

7.6.5 Miss Rate Improvement

7.6.5.1 Effect on NVM based Main Cache

Due to hits in the victim cache, the miss rate gain by the proposed technique: WLVC over the Base STT-RAM and SRAM are 23.1% and 23.8%, respectively. Over RWHCA, due to large data placement in the small-sized SRAM region of an existing technique, WLVC gets a miss rate improvement of 31.8%. These respective improvements can be seen from fig. 7.17. However, we get a marginal

Policy	O	P	Q	S	row
RWHCA	6.55%	5.96%	7.42%	7.81%	1
Base HCA	0%	-0.6%	0.87%	1.26%	2
Base STT	0.3%	-0.31%	1.16%	1.55%	3
Base SRAM	0.31%	-0.33%	1.18%	1.57%	4

TABLE 7.11: Percentage improvement in miss rate (higher is better)

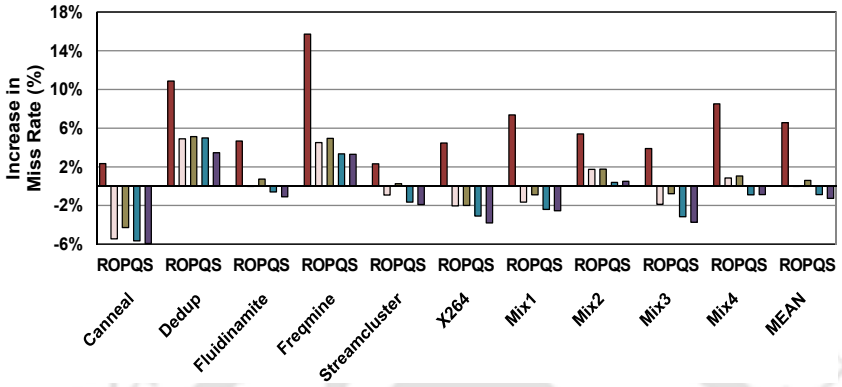


FIGURE 7.18: Percentage increase in miss rate for R, O, P, Q and S against Base STT (lower is better)

improvement of the miss rate in the proposed technique with respect to VC, which is a technique similar to the proposed technique in terms of miss rate.

7.6.5.2 Effect on HCA based Main Cache

The miss rate improvement by the proposed techniques against the base STT is given in figure 7.18. Section 7.6.5.2 reports the improvement percentages in miss rate by the proposed varieties against the base STT, SRAM, HCA, and RWHCA. We observe an improvement of 7.81% in the miss rate with the existing technique RWHCA (row-1) due to the applied victim cache and its proposed optimized policy. However, the proposed technique maintains the same miss rate with all baselines that shows the effect of associating the victim cache with the main hybrid cache. In other words, we can have policies to reduce writes in STT (e.g., RWHCA) and use VC to make up for degradation. Besides, our intelligent block movement policies also control the write endurance of HCA.

Metrics	Policy	O	P	Q	S
Writes Savings	SRAM 2MB	27.6%	27.5%	27.9%	27.8%
	STT 6MB	-4.98%	-5%	-4.8%	-4.9%
CPI Gain	SRAM 2MB	4.38%	4.42%	4.72%	5.34%
	STT 6MB	1.44%	1.48%	1.8%	2.43%
Exec. Time Gain	SRAM 2MB	4.47%	4.5%	4.8%	5.3%
	STT 6MB	1.56%	1.58%	1.85%	2.43%
Energy Overhead	SRAM 2MB	8.7%	8.6%	8.3%	7.73%
	STT 6MB	5.95%	5.9%	5.63%	5%

TABLE 7.12: Iso area analysis between proposed HCAs and base SRAM and STT

7.6.6 Iso Area Analysis for HCA based Main Cache

Along with the detailed iso-capacity analysis, we have also presented an iso-area analysis. In particular, within the same area footprint of proposed hybrid caches, the analyses with different sizes of SRAM and STT are reported in table 7.12. Note that the negative values in the table imply an increase in the metrics values. During the analysis, we have assumed STT is three times denser than SRAM, and henceforth, for 4MB hybrid caches (3MB STT and 1MB SRAM), we can accommodate 2MB SRAM and 6MB STT with the same area footprint. Thus, the results obtained by iso-area analyses indicate that there is a significant gain in performance with marginal energy and write overhead.

7.6.7 Storage and Area Overhead

In WLVC, along with the victim cache, we incorporated 2-bit read and write counters with every block of the main cache and a 4 and 5-bits weight and timestamp with every block of the victim cache. In addition to this, we add two 42-bit swap buffers. All these constitute the storage overhead percentage of 0.771% over Base STT-RAM for a 32-entry victim cache. The area overhead by the proposed design concerning Base STT is 21.7%.

Whereas, the techniques with HCA, we incorporate a single bit *r_bit* with each entry of victim cache along with that two 42-bit swap buffers are used to facilitate the tag swapping process between the main hybrid cache and the victim cache. In addition, we used two 12-bit counters and four 5-bit counters to maintain the

Cache Configuration	Reference Policy	CPI Gain (%)	Exe. Time Gain (%)	Miss Rate Gain (%)	EDP Gain (%)
M = 1MB	**STT**	12.04%	12.27%	46.4%	-9.11%
	STT+VC	5.55%	7.57%	4.02%	8.24%
M = 2MB	**STT**	8.70%	8.50%	43.50%	-7.38%
	STT+VC	3%	4.50%	1.71%	3.82%
M = 4 MB	**STT**	4.63%	4.80%	43.45%	-3.47%
	STT+VC	2.32%	2.56%	0.98%	3.20%
M = 8MB	**STT**	3.53%	3.92%	43.4%	-2.61%
	STT+VC	1.41%	1.44%	0.01%	1.83%

TABLE 7.13: Comparative analysis for different capacity of NVM LLC (M)

LLC size	CPI Gain (%)	Write Access Gain (%)	Exe. Time Gain (%)	Miss Rate Gain (%)	Energy Gain (%)
1MB	9.87%	-2.6%	9.46%	3.33%	37.3%
2MB	7.61%	-1.39%	7.17%	1.82%	37.1%
4MB	4.43%	-1.8%	4.18%	1.55%	34.1%
8MB	2.8%	-0.84%	2.5%	0.35%	32.1%

TABLE 7.14: Comparative analysis for different Hybrid LLC capacity

eviction and the count of the victim region in RDVCP. All these constitute the storage overhead percentage of 0.05% with respect to Base pure STT (having no victim cache associated) for 32 entry victim cache. On the other hand, the area overhead percentage with respect to Base pure STT is 0.35% and with respect to 4MB 16-way set associative Base-HCA (contain 12-way STT and 4-way SRAM) with no victim cache is 0.18%.

7.7 Parameter Comparative Analysis

In addition to the results presented in the previous section, we also experimented with different configurations of main LLC STT/HCA and victim cache and, with varying values for the parameters of proposed technique: WLVC and RDVCP. In this section, we show the effect of various metrics in comparison to the chosen parameters. The values are given against the Base STT.

VC Configuration	Reference Policy	CPI Gain (%)	Exe. Time Gain (%)	Miss Rate Gain (%)	EDP Gain (%)
V_n = 16 entries	STT	3.35%	3.20%	42.6%	-5.1%
	STT+VC	1.6%	1.43%	0.73%	2.20%
V_n = 32 entries	STT	4.63%	4.80%	43.4%	-3.47%
	STT+VC	2.32%	2.57%	0.98%	3.20%
V_n = 64 entries	STT	5.10%	5.00%	44.5%	-3.96%
	STT+VC	2.00%	2.21%	2.2%	1.88%

TABLE 7.15: Comparative analysis for different victim cache sizes (V_n) for NVM cache

Victim size (V_n)	CPI Gain (%)	Write Access Gain (%)	Exe. Time Gain (%)	Miss Rate Gain (%)	Energy Gain (%)
16 Entries	4.1%	-1.6%	3.83%	0.79%	34.9%
32 Entries	4.43%	-1.8%	4.18%	1.55%	34.1%
64 Entries	4.83%	-1.87%	4.85%	2.93%	33.1%

TABLE 7.16: Comparative analysis for different victim cache sizes (V_n) for HCA

7.7.1 Change in LLC size

Tables 7.13 and 7.14 shows the comparative analysis between different cache capacities of the main cache associated with the victim cache. The tables show gains obtained by the use of proposed policies over the reference policy: STT (pure STT-RAM main cache) and STT+VC (STT-RAM main cache with VC simple cache in case of WLVC). Cache capacity impacts the residency of the block. Specifically, smaller sized cache suffers from large capacity miss as compared to larger sized cache. As can be seen from the tables, the victim cache associated with a small cache would be more beneficial as compared to the large cache. The reason is that with a small cache, the victim cache holds blocks that are evicted prematurely without finishing their lifetime. Further, by incorporating our techniques, gain in the performance is observed. Thus, the embedded system having a small capacity main cache can benefit from this proposal. Note that the negative value implies the increase in metrics values.

7.7.2 Change in Number of VC entries (V_n)

Impact in metric value with different sizes of victim cache sizes is shown in tables 7.15 and 7.16. The number of victim cache entries impacts the residency of

Thr	CPI Gain (%)	Exe. Time Gain (%)	EDP Gain (%)	Miss Rate Gain (%)
T=0	5.2%	7.05%	-5.35%	41.5%
T=1	4.63%	4.8%	-3.47%	43.45%
T=2	3.70%	3.5%	-1.52%	44%

TABLE 7.17: Comparative analysis for different Bias (T) for NVM main cache

the block in the victim cache. Larger victim cache gives good performance gain and improvement in miss rate. However, the larger search time of fully associative victim cache impacts the CPI gain and the EDP/Energy over the chosen victim entries (32). On the other hand, smaller victim cache degrades the performance with respect to STT-RAM as well as miss rate. Also, our technique gets a performance improvement with respect to STT with victim cache (STT+VC). Thus, a careful selection of victim cache size will make the hardware efficient. Note that the negative value implies the increase in metrics values.

7.7.3 Change in Bias

7.7.3.1 Impact on NVM based Main Cache

Table 7.17 presents the impact of varying the *Bias* on different metrics value for WLVC. Change of *Bias* impacts the decision of directly serving of a block from the victim cache upon a hit. With smaller *Bias* value, the performance of the cache is better as a large number of write-intensive blocks are served from the victim cache itself with less number of swaps compared to reference case. However, this impacts the EDP of the cache, with an increase in the miss rate, because the VC is not able to store several entries. On the other hand, with the large *Bias*, the performance impact is smaller compared to the reference case as the main cache itself handles most of the writes.

7.7.3.2 Impact on HCA based Main Cache

The change of bias with different metric values is reported in table 7.18. Change of Bias affects the change of victim region partition size. The smaller bias value

Bias	CPI Gain (%)	Write Access Gain (%)	Exe. Time Gain (%)	Miss Rate Gain (%)	Energy Gain (%)
100	3.34%	-1.54%	3.69%	1.07%	34.4%
250	3.98%	-2.4%	4.01%	1.26%	33.8%
500	4.43%	-1.8%	4.18%	1.55%	34.1%
1000	4.01%	-2.61%	3.31%	0.97%	32.6%

TABLE 7.18: Comparative analysis for different Bias for HCA main cache

Interval (I)	CPI Imp. (%)	Write Access Imp. (%)	Exe. Time Imp. (%)	Miss Rate Imp. (%)	Energy Imp. (%)
I=0.5M	4.57%	-2.7%	4.34%	2.01%	32.8%
I=1M	4.43%	-1.8%	4.18%	1.55%	34.1%
I=2M	3.91%	-2.42%	4.07%	0.78%	33.8%

TABLE 7.19: Comparative analysis for different interval (I) for HCA main cache

results in a frequent change in partition size and thereby creates instability in the different regions of the victim cache (along with lesser CPI and execution time improvement). On the other hand, large bias value causes a less frequent reconfiguration of VC partition. This results in a large number of eviction from the SRAM partition of the main hybrid cache, which is not able to remain in VC, leading to an increase in the miss rate.

7.7.4 Change in Interval (I)

The impact for the change in the interval is presented only for the hybrid cache as we have not got much difference in the metric value for NVM based main cache. Table 7.19 shows the comparative analysis for distinct interval values. The interval values affect the frequency of change in the partition size of the victim cache for the different regions of the main hybrid cache. With a large interval, the frequency of change in the partition is smaller, thereby increases the eviction from the SRAM partition of the main hybrid and reduces the miss rate improvement. On the other hand, smaller interval results in the frequent change in the partition size, which creates instability in the different regions of the victim cache thereby increasing the number of write accesses and energy in HCA.

Thus, the careful selection of interval (I) and *Bias* values will result in more efficient use of victim cache.

7.8 Conclusion

This chapter exploits the impact of associating victim cache with the NVM/HCA based main cache. The victim cache aims to reduce the miss penalty by placing the victims from the main cache in a fully associative SRAM based structure. In NVM based main cache architecture, with each hit in the victim cache, a block interchange is required between the main cache and victim cache. To save time and energy due to these interchanges, we proposed a policy that caches the write-intensive block inside the victim cache upon hits. These blocks are served from the victim cache itself. Also, in victim cache, we have used Weighted Least Recently Used (WLRU) Replacement Policy to evict the victim. In WLRU, the eviction decision is based on both weight and the last accessed timestamp.

Whereas, in HCA, the victim cache is used to retain the evicted blocks from the main cache. With each miss in the main hybrid cache, the victim cache is searched. Upon a hit in the victim cache, we proposed a policy that effectively places the block to the appropriate region of the hybrid cache based on the type of request and the victim block's dirty bit status. We also proposed a dynamic region-based victim cache partition technique to manage the run time load and uneven evictions from the SRAM region of the main hybrid cache. The partitioning technique enables the victim cache to hold the victim dedicated to each region thereby it increases the possibility of caching the most recently used blocks evicted from the SRAM as well as STT partition of the main hybrid cache. Note that the main aim of the proposed partitioning technique is to improve the efficacy of the victim cache to store the appropriate number of blocks from each region.

To measure the efficacy of the proposed technique, we compared our proposed techniques with one of the existing HCA techniques and with different baselines. Experimental evaluation on a full system simulator shows that by associating

victim cache with NVM/HCA based main cache, the proposals get significant performance gains for different last level cache sizes. Thus, the effective use of victim cache with the main hybrid cache aids in improving the performance and makes the hardware overall efficient.

भारतीय प्रौद्योगिकी संस्थान गुवाहाटी
Indian Institute of Technology Guwahati

Chapter 8

Conclusion

This research work is motivated towards improving the utilization of emerging non-volatile memory technologies and make them as a viable option for the last level caches. Towards this, we worked in the following two directions: (i) to overcome the costly write operations and improve the performance and reduce the energy consumption, (ii) to extend the lifetime of the NVM caches affected due to the weak write endurance and unwanted write variations. Towards proposing a solution for the former direction, we make use of the concept of hybrid cache architecture where a large portion of NVM is integrated with a small portion of SRAM. In this context, two techniques were proposed in this dissertation that make use of prediction and private blocks; and utilizing a victim cache. Whereas, for proposing a solution for the latter direction, we proposed two kinds of wear leveling techniques: Intra-set wear leveling (methods to minimize the write variation inside the cache set) and Inter-set wear leveling (techniques to reduce the write variation across the cache sets). This chapter sums up all the proposed contribution of this dissertation along with the future directions for research.

8.1 Summary of Contributions

- **Block Placement and Replacement Method to Counter Costly Write Operations:** To overcome the costly write operations, we used the concept

of Hybrid Cache Architecture (HCA). In HCA, block placement is a critical task as it is expected from the SRAM region to handle the writing pressure of the applications; thereby, saving the NVM region from entertaining the writes. Towards this, we reduce the number of writes in the NVM portion of the HCA by using the concept of private blocks. In particular, we identified those blocks that are exclusively accessed by higher-level caches and allocate these blocks to the NVM portion as dataless entries. Additionally, to further save the writes in the NVM portion of the cache, a Reuse Distance Aware Write Intensity Predictor (RDAWIP) was employed in our proposal. The RDAWIP predicts the reuse distance aware write intensity behavior of the cache block and appropriately redirects the write-backs from L1 cache for the dataless entries of the NVM portion to the SRAM region of L2 hybrid cache. Besides the prediction and the placement of the block, we made use of different fields of the predictor in block replacement decision. The replacement decision is inspired by the reuse distance and the access behavior of the cache block. In a quad-core system, for an 8MB 16 way set associative cache, the proposed method saves the number of writes by 34.5% with 56.3% gain in energy, while maintaining the performance.

- **Methods to mitigate the intra-set write variation:** In this proposal, four intra-set wear leveling techniques were proposed that work on the basic concept of write restriction. These four methods works at the different granularities of the cache bank: (1) Partitioning the cache vertically into multiple equal-sized windows (one window comprises of multiple ways) (2) Partitioning the cache horizontally into multiple equal-sized modules (one module consist of multiple sets) (3) Way Level Granularity. The intra-set write variation is controlled by selecting different window/ways/sub-ways as the read-only over different execution intervals. The window/way/sub-ways selection is made using i) round-robin method or (ii) write counts for each way (inside the cache) / sub-way (inside the module). In a quad-core system, for a 16MB 16 way set associative cache, the proposed methods reduce the intra-set write variation in the range of 80%-86.5% and improve the lifetime by 7.27 times over baseline and 5.86 over existing technique, while maintaining the performance.

- **Methods to mitigate the inter-set write variation:** In this work, two techniques were introduced that were based on fellow sets and Dynamic Associativity Management. In our proposals, sets are logically grouped into different fellow sets. Each cache set in the group is divided into Normal (NP) and Reserve part (RP). Within each fellow group, to control the write variation across the cache set, the heavily written sets spread out their writes to the RP of the lightly written cache set. Two variations of the approach were designed (based on the static and dynamic RP) and evaluated. In quad-core system, for 8 MB 16 way set associative cache, the proposed method reduces the inter-set write variation in the range of 27.6% - 34% and improves the lifetime by 14.7 - 20.7% over baseline and 6.58 - 12.11% over existing policies.

- **Methods to improve the performance:** In this proposal, to improve the performance degraded due to costly write latency for NVM cache and the increased miss rate for HCA, the victim cache was used. The integration of victim cache with NVM cache requires an exchange of block between the main cache and the victim cache. To save on the time to exchange and the subsequent slow writes to these blocks in the NVM cache, we move only non-write intensive blocks to the main cache based on their write intensity. Whereas, hits for write-intensive blocks are served directly from the victim cache. In a quad-core system, for 4MB 16 way associative cache, the proposed approach gets a performance improvement of 5.88% over STT-RAM and 5.43% gain in execution time with the marginal energy consumption. On the other hand, the employment of victim cache with hybrid cache needs an effective block placement policy to place the block into the appropriate region upon victim cache hit. An access based block placement technique is presented in this context where the block is placed based upon the type of access. Besides, to manage the run time load and the uneven evictions from the SRAM partition, we gave a dynamic region-based victim cache partitioning method to hold the victims dedicated to each region. In quad-core system, for 4MB 16 way set associative cache, the proposed method gets a performance improvement of 4.43% and 4.18% gain in execution time with the reduction in 7.81% in miss rate over the existing HCA policy.

Figure 8.1 summarizes the contributions of this thesis.

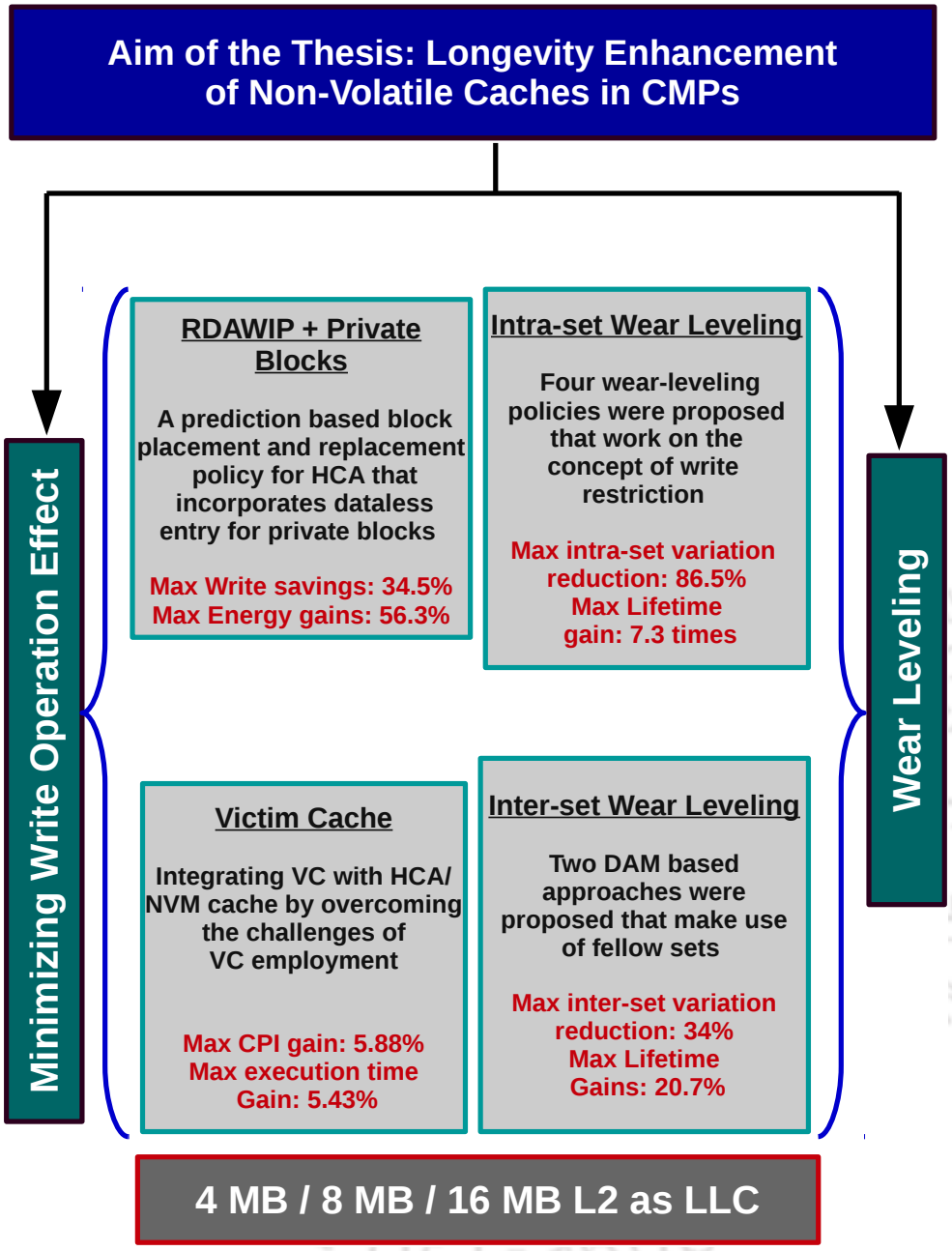


FIGURE 8.1: Summary of the thesis contributions. The results are shown for different capacities (4MB, 8MB and 16MB) of LLCs. Note that, the proposed architectures are also evaluated with various cache configurations and with different prior works.

8.2 Scope for Future Work

The contributions of this thesis can be extended in several ways. Some of these possible future research directions are listed below:

- For each presented proposals in this thesis, there are some changes needed in the functionality of the cache controller. In the future, all the changes required will be modeled, and the extra power, timing, and area will be considered to measure the actual efficacy of each proposal.
- Our proposed intra-set wear-leveling techniques reduce the significant amount of write variation at the different granularity of cache bank. In the future, the write restriction can further be explored at the level of a cache blocks and memory cells to reduce the unwanted write variation.
- The fellow set based DAM inter-set wear leveling approach works well to improve the lifetime. But, the fellow group established by the proposals are having the fixed location cache sets. Considering the dynamic behavior of the application, different fellow groups having different write pattern behavior. In particular, some of the fellow groups are lightly written, and some of them are heavily written. By considering this fact, an appropriate dynamic mapping policy can be proposed that forms the fellow groups with different locations of cache set rather than deterministic location of cache sets.
- Victim Cache can further be attached with the NVM/HCA based L1 cache to enhance its performance. Basically, towards applying this, we need an effective method that takes care of the various challenges to employ victim cache with NVM based L1 cache to improve the performance of the system.

भारतीय प्रौद्योगिकी संस्थान गुवाहाटी
Indian Institute of Technology Guwahati

Appendix A

Simulation Framework

This appendix focuses on the experimental setup used in our approaches. The entire experiments given in this dissertation are performed on a full-system simulation framework. Fundamentally, the full system simulators model the electronic system as a whole, including CMPs. The system on which the full system simulator executes is called as the *host system* and the virtual environment setup by the full system framework is called as *target system*. Additionally, the full system framework develops the flavors of CMPs by including the multiple modules for the CPU cores, the multi-level private or shared cache hierarchies, memory controllers and system, and I/O devices. These multiple small systems are all together connected and communicated by the well known Network-on-Chip (NoC) module that has been attached in the full system framework. Contrary to the instruction set simulator, the full system simulator allows the applications to execute independently through an Operation System (OS) installed on the target system. Also, the target system allows the kernel modules of the OS to run through the virtual drivers as normally on a real hardware system.

Precisely, simulators are the set of computer programs that run on the host systems; thus, any functionality developed for the target system can be easily altered to meet the new design requirement. For instance, the conventional cache made up of SRAM modeled on a full system framework can be changed to NVM cache or the hybrid cache at both banks as well as way based granularity. For design

space investigation in the target system, the basic preliminary parameters like cache size, associativity, number of banks, etc. can be easily altered. The brief history of computer architecture simulators is discussed in the next section with their importance in academic and industrial research.

A.1 Computer Architecture Simulators

A computer architecture simulator or an architecture simulator is a set of the software programs that are used to investigate the power and performance of any modeled computer system. The modeled system can be either (a) full system that simulates the complete computing system, or, (b) target microprocessor called instruction set simulator. Amid, the various simulation techniques, discrete event simulation, and trace-driven simulators/emulators are commonly used by the computer architects [125]. As the emulators are simulators with hardware design constraints, thus, we choose to use discrete event and trace-driven simulators in our experimental analysis.

In the real time fabrication, the actual imitation of modern CMPs is complicated [126], expensive and highly integrated as it contains the number of cores, multi-level cache hierarchies of different sizes, memory controllers, NoC, etc. on a single wafer real-estate. Additionally, for the experimental analysis, altering the various design parameters of the architecture before the deployment for the same is required. For instance, while developing any wear-leveling technique, we need a heavyweight result analysis that can generate different design choices for the next generation computing system. In this context, once the approach is implemented in the simulator, we can conduct various experiments for different cache associativity, different cache sizes, and with the different parameters used in the method. On real hardware, the prototyping of each architecture with different design space exploration is impractical in the academic research environment [127, 118]. Furthermore, the real hardware prototype built by using Field Programmable Gate Arrays (FPGA) does not have user-friendly debugging space. The costlier tools

and the complex designing process for a single academic project comprising of multiple small design proposals are not always possible. For instance, in this dissertation, the multiple wear-leveling techniques: SWWR, DWWR, DWAWR, FSSRP, FSDRP, Polf, Swap-Shift are designed with the in-depth lifetime and the write variation analysis on different cache sizes and associativity. Thus, the computer architects use the simulators to conduct their experiments in a timely and cost-effective manner and to evaluate the proposed designs in the context of power and performance over the existing ones [126, 128]. Maximum of the listed state of the art works illustrated in Chapter 2 are implemented and evaluated using full system simulators.

The oldest and the first machine simulator, SimOS [129], simulates the machine hardware through the service given by the underlying operating system. Later, the extensively known architectural simulator, SimpleScalar [130] has been developed to model the set of super-scalar processors. Even though SimpleScalar is widely accepted, it does not support the multi-core systems. Soon afterward, the multiple simulators have been designed for the fault analysis [131], event-driven simulation [132], and verification tool for micro-processor based on virtual machines [133]. As the simulator runs with the help of the host system, the performance of the simulator will heavily depend upon the performance of the host machine. In particular, if the throughput of real hardware increases, the full system cab be modeled without affecting performance [128]. In addition, some of the simulators have immanently sequential nature that execute slower than the ones which have inherently parallelized environments. Over the previous years, many simulators have developed for various needs and requirements [126, 118, 131, 132, 133, 134, 135, 136].

The full system simulators can be classified into two types: Timing and Functional [127, 118]. As the name suggests, the functional simulator mostly depicts the real functionalities or the activities of an actual system. Whereas, the timing based full system simulator models the real-time behavior of the system that follows a discrete event simulation technique. Furthermore, it imitates the functionality of an actual system with the timings on which the task has to be triggered.

Generally, the timing simulation is needed for comparing the LLC performance in modern CMPs.

A.1.1 GEM-5

The gem5 [118] simulation framework is combination of the best features of M5 [128] and GEMS [127] simulation tools. The M5 provides the diverse CPU models and multiple Instruction Set Architectures (ISA), whereas, the GEMS augmented detailed and flexible memory system with the support of cache coherence protocols and the complete interconnection network. This subsection briefly discusses the features and the modules of M5 and GEMS along with their limitations.

A.1.1.1 M5

The M5 is a full system execution driven simulator that generates the complete target system or a virtual machine which runs on top of a host system. M5 simulator is an open-source and acts as an alternative to the commercial Simics simulator. It was initially intended/developed to analyze the throughput of interconnect and network protocols by modeling multiple client-server machines. Furthermore, it is flexible enough to model and support diverse CPU models, including in-order and out-of-order cores with the capability of both memory, I/O and OS development. Besides this, M5 assists by supporting different ISAs, such as ALPHA, ARM, MIPS, Power, SPARC, and X86. M5 is fast enough to execute realistic benchmark suites, like SPECWEB99 [137], Netperf [138], Surge, iSCSI, etc.

The main aim of architectural research is to model the next-generation architecture to support the ever changing/variable computing system. In this dissertation, we propose emerging NVM architectures that cope up with the increasing data demands. The full system framework provided by the M5 is suitable to design such future memory architecture without any physical overheads. Also, with respect to the commercial industry, the designing and the verification of such emerging memories in M5 with the constraints of limited time and space is also useful

enough. Moreover, the development of the software for future hardware can be done parallelly as the simulated framework provides the taste of the whole system.

A.1.1.2 Restrictions in M5

M5 has a powerful capabilities, still it lacks behind to model detailed and flexible memory system supporting multiple coherence protocols and interconnection network that are needed to simulate the CMPs. In particular, M5 simulates only the point to point snooping based interconnection and caches that are not scalable and flexible for the CMPs. GEMS [127], a timed simulator has been proposed and that is merged and works on top of M5. The purpose of designing the GEMS is to model the complete memory hierarchy of CMPs with coherence management and on-chip communications through the Network-on-Chip (NoC). The timing simulation feature provided by GEMS helps to evaluate different CMP architectures. The brief details on GEMS are provided in the next sections. GEMS cannot work alone without the M5, hence, the functional behavior is disjoint with the timing models.

A.1.1.3 GEMS

GEMS [127] has two major modules: Ruby and Garnet [139]. Ruby simulates the complete memory hierarchy of CMPs comprising of L1 cache, L2 cache, memory banks, directories, etc. Each component of Ruby is called as "Machine" and is identified by its unique ID: called MachineID. The on-chip communication between these machines is determined by their MachineIDs and is through the underlying NoC, managed by Garnet. The CMP based architecture model in Ruby uses Garnet for providing on-chip communications. Garnet models the real-time events for transferring packets through the NoC. Additionally, Garnet also models a variety of network topologies for NoC with different design options.

The block fetch request from the M5 processor is passed to the Ruby modules of GEMS. For the fetch request, the very first level of cache,i.e., L1 detects the

miss or hit. If the block is found in the first level, the M5 core continues its execution. Otherwise, the block request from the generated core is stalled until GEMS completes its simulation for the miss. The timing dependent functional simulation is controlled and driven by Ruby.

With each L1, a sequencer is incorporated that manages the request from the corresponding core. In CMPs, there will be multiple levels of caches that deals the cache requests simultaneously. With each level of cache, a controller is attached that performs all the functionalities related to the cache by taking into consideration consistency and persistence of shared data. The domain-specific language called SLICC is used to manage the modeling of such controllers. In particular, SLICC manages all the operations and the communications of the controller with the other machines in the system. Other than that, one of the biggest concern with the CMP based cache structure is to manage the coherence in the shared cache. Multiple coherence protocols have been implemented in GEMS that are governed by the different modules of controllers. These controllers are communicate through message passing, which is supported by the NoC. The designing of the protocol is the responsibility of SLICC as it is the combined duty of the controllers.

A.1.1.4 CMP Architecture Supported by GEMS

GEMS supports SNUCA based cache architecture, which is very robust and can be configured with a different set of features like cache size, number of banks, hit time, miss penalty, access latency, etc. These parameters can be easily modified by altering the configuration file. Other than these, some additional parameters can be set based on the architecture demand, such as block-size, number of virtual networks, replacement policy, cache associativity, etc. In each experiment for the baseline and the proposed architectures, we use the MESI protocol, which is termed as "MESI-CMP" protocol in GEMS.

GEMS models the baseline cache architecture where the read and write latency to perform the respective operations are same. In our work, to implement the

asymmetric access latencies for NVMs and hybrid caches, we use the concept of bank contention. Bank contention allows the respective bank to entertain only one access request at a time. To facilitate this process, we made some changes in the protocol level to manage the coherence. In particular, the programs written in the SLICC has been modified to support the proposed architectures. Also, to support the proposed designs that run on top of the baseline NVMs, other internal structures of the memory system have been altered. For instance, to implement the Victim Cache, a new cache structure has been added with the existing cache memory. To validate this, rigorous testing has been done to guarantee the correctness of the proposed work. At last, compilation of GEMS builds the new architecture design.

A.1.1.5 Result Analysis

As GEM5 is a full system simulator, it can run a real set of applications on the simulated architecture. During the experiments, GEM5 logs the various statistics of the running application. Some of the important information needed in this dissertation are described below:

- Total Cycle Executed: The metric records the summation of all the cycles executed for all cores. Besides this, GEM5 also collects the executed cycles (comprising of busy and idle cycles) for the individual core.
- Total Simulated Instruction: This collects the each cores number of executed instructions as well as outputs the summation of total instructions executed.
- L1 Demand Access: GEM5 also records the demand accesses including demand hit and miss for each L1 bank private to a core.
- L2 Demand Access: It records the individual demand hit and misses for each shared L2 bank.

Other than that, we have also added some of the additional metrics that are needed to analyze the conducted simulations for the hybrid and NVM caches:

- NVM Read/Write Hits: The metrics are needed for the hybrid cache. It collects the number of reads, and the writes incurred in the NVM portion.
- SRAM Read/Write Hits: This implies the same as the NVM hits, but for the SRAM portion of the hybrid cache.
- InterV/IntraV: The metric records the coefficient of write variations (as given in Equations 2.1 and 2.2) present in the non-volatile cache bank.
- Cache Block Write: The metric outputs the number of individual writes for each cache block. It is used to calculate the lifetime improvement (as given in Equations 2.3 and 2.4) by the proposed approaches.

Apart from these metrics, the other metrics like Cycle Per Instruction (CPI), Instruction Per Cycle (IPC), Miss Per Thousand Kilo Instructions (MPKI), etc. are easily derived or calculated from the given documented metrics provided by GEM5.

A.1.1.6 GEM-5 Limitations

With many unique features and different types of ISA and CPU support, there are some limitations associated with GEM-5, which are as follows:

- GEM-5 is unsuitable for memory and cluster exploration [140]. GEM-5 assumes that there is an interface available from each tile to the main memory. As a result, only the access latency to the memory is captured but not the actual traffic from the directory/LLC to the memory.
- Several previous studies [141, 142, 143] pointed out the inaccuracies in the simulation results over different ISAs by the GEM-5. These inaccuracies are due to imprecise decoding of instructions into micro-operations, high cache misses, and over-estimated branch misprediction [144].

A.1.2 Timing and Power Modeling Tools: CACTI and NVSIM

The GEM5 full system framework simulates many real sets of applications by utilizing the underlying architecture. However, this framework is not able to model the timing, power, and area for different memory technologies and at different cache level granularities. CACTI 6.5 [19] and NVSIM [21] are two well known simulators in the computer architecture research community, that take some of the architectural parameters like - cache memory technology, cache size, cache associativity, cache level, block size, access technique (UCA or NUCA), etc to simulate the cache at device/circuit level. CACTI 6.5 is responsible for modeling the traditional SRAM caches, whereas, NVSIM models different emerging NVM memory technologies like STT-RAM, ReRAM, PCRAM, and the NAND flash. By modeling the architecture at the device level, the simulators output the power consumption, area overheads, cache access time, etc. Based upon the ITRS reports [10, 145, 146], the NVM and the SRAM memory technology fabricated caches can be of three categories based on the power and performance modes: (a) HP: known as High Performance cell that consumes large power and very fast in the access operation; (b) LSTP: Low STandby Power cells, incurs low power when idle. However, their accessing is slower than the HP as the transition from the low power standby mode to active mode incurs extra cycles; (c) LOP, known as the Low Operating Power cells that incurs less power both in standby as well as active mode. It is the slowest among the three methods. Among these modes, we have chosen the HP mode for the cache construction in our work. Furthermore, CACTI and NVSIM support three types of cache access techniques: fast, sequential, and normal. In our work, we have used the fast mode of accessing the cache. In fast mode, both data and tag arrays are searched concurrently to identify hit or miss. For calculating the power consumption, the transistor length plays a prominent role here, and it is also known as the technology parameter in CACTI and NVSIM. In our work, we have used the transistor of the channel length of 32nm and the temperature of 350K. Internally, NVSIM uses empirical modeling methodology

of CACTI, but it also includes several other new features which are described as below:

- It facilitates the user to model their memory cell configuration.
- It gives the various design optimization options for buffer like latency, area and balance optimization.
- It facilitates the user to model the memory banks in a bus like structure rather than only in the H-Tree structure.
- It models the various types of data sensing schemes rather than only the voltage based sensing.

A.2 Benchmarks

As reported in the previous section, the full system simulator like GEM5 runs real benchmarks on a simulated architecture. Based on the stats collected from these simulations, the performance of the architecture is evaluated. The power and the performance results of the simulated architecture give enough idea to the researcher and the hardware manufacturer about the real-world behavior of the new architecture design. To facilitate this process, several benchmarks suites like PARSEC [6], SPEC CPU 2006 [7], SPLASH-2 [147], etc. are available in the market. In this dissertation, we have used multi-threaded PARSEC benchmarks and multi-programmed SPEC CPU 2006 benchmark suite to test our architecture design. The detailed description of these benchmark suites are given below:

In this dissertation, we have not considered the SPLASH-2 benchmark for the evaluation as their small input size cannot be used for the large-sized LLC.

A.2.1 PARSEC

The Princeton Application Repository for Shared-Memory Computers (PARSEC) [6], a benchmark suite made up of the next generation multi-threaded applications. The PARSEC benchmarks are developed for the evaluation and the validation of the next generation CMPs. It is the collaborative project between Princeton University and Intel to develop such benchmark applications that help the research community for efficient design of future computing systems. The PARSEC benchmark suite is open-source and is widely accepted in the architecture research community. It is used in both academic as well as industrial research. Some of the essential objectives of PARSEC are given below:

- Next generation applications for different real-world problems.
- Distinct input size for each and every application.
- Focus on multi-threaded applications.

The benchmarks that were used before PARSEC are application-specific and are executed in the serial fashion [6]. PARSEC version 2.1 has 12 applications, and each application is parallelized and multi-threaded. These applications are selected from diverse real-world areas like computer vision, animation physics, finance, media processing, etc. The detailed description of PARSEC benchmarks is given in table A.1 [6]. Typically, the multi-threaded application exchanges data between its spawned threads. The data sharing and exchange description of these benchmarks are reported in table A.2 [6].

The term benchmarks are also called alternatively as workload, application or program.

Each benchmarks in PARSEC has its own working set with different input sizes: large, medium, small, etc. Based upon the requirement and their architecture design, users can run workloads with any size of input set.

Program/ Benchmarks	Application Domain	Parallelization		Working-Set
		Model	Granularity	
blackscholes	Financial Analysis	data-parallel	coarse	small
bodytrack	Computer Vision	data-parallel	medium	medium
canneal	Engineering	unstructured	fine	unbounded
dedup	Enterprise Storage	pipeline	medium	unbounded
facesim	Animation	data-parallel	coarse	large
ferret	Similarity Search	pipeline	medium	unbounded
fluidanimate	Animation	data-parallel	fine	large
freqmine	Data Mining	data-parallel	medium	unbounded
streamcluster	Data Mining	data-parallel	medium	medium
swaptions	Financial Analysis	data-parallel	coarse	medium
vips	Media Processing	data-parallel	coarse	medium
x264	Media Processing	pipeline	coarse	medium

TABLE A.1: The inherent key characteristics of PARSEC benchmarks

Program/ Benchmarks	Data Usage	
	Sharing	Exchange
blackscholes, swaptions	low	low
bodytrack, freqmine	high	medium
canneal, dedup, ferret, x264	high	high
facesim, fluidanimate, streamcluster, vips	low	medium

TABLE A.2: The data usage behavior of PARSEC benchmarks

A.2.1.1 Benchmark Descriptions

This section illustrates the properties of few PARSEC benchmarks that are used to evaluated our proposed architecture designs. The detailed content for the rest of the PARSEC benchmarks is reported in [6].

Bodytrack: The body track application records the 3D view of the human body through various cameras. An annealed practice filter is used to capture the 3D view using foreground and edge silhouette. In this application, the input video that contains many frames is used to select as reference frame. Different frames at different time-stamp are selected, and the likelihood value is computed with the reference frame. The likelihood is a degree of the 3D body alignment with its foreground and its edges in the image frame. The likelihood value is calculated

by considering two attributes: the foreground and the edge distance map. The benchmark has a persistent thread pool where the main thread assigns the task to the thread pool. Before proceeding further, the main thread has to wait for the remaining threads to complete their execution.

Fluidanimate: Due to the increasing importance of physical simulation and real-time animation of computer games, the fluidanimate application is included in the PARSEC benchmark suite. It is an Intel RMS application that uses Smoothed Particle Hydrodynamic method [148]. For modeling the incompressible fluid for interactive animation, fluidanimate uses five kernels. The application produces output based on interpreting and discovering the surface of thick fluid.

Freqmine: It is an Intel RMS application developed by Concordia University. The reason for the inclusion of freqmine workload in the PARSEC suite is the increasing demand for data mining techniques. The freqmine application is used for Frequent Itemset MIning (FIMI) [149] with an array-based version of frequent pattern growth. The FIMI is the foundation of Association Rule Mining (ARM), which is a common data mining problem for areas like market data, log analysis, protein sequence, etc. The application uses three kernels and is parallelized with OpenMP.

Swaption: It is an Intel RMS workload that is used for pricing the portfolio by using the Heath-Jarrow-Morton (HJM) [150] method. Due to increased importance of Partial Differential Equation (PDE) and the Monte Carlo Simulation, the Swaption workload is added into the PARSEC benchmarks. In Swaption, the behavior of the HJM model is non-Markovian, which prevents the solving of PDE for the computation of price. Thus, the application uses the Monte Carlo Simulation. The application stores all the portfolio in the swaption array, where each of the array entry represents a derivative. The array is further divided into the number of blocks that is same as the number of spawned threads. Hence, to ensure

parallelism, each block is assigned to a thread. To compute a price, the swaption application iterates through all the blocks and calls the module HJM Swaption blocking.

Canneal: The canneal workload uses the cache-aware Simulated Annealing (SA) technique to minimize the routing cost of chip design. SA is a well-known method to approximate the global optimum in a large search space. The canneal application randomly chooses two pairs of elements and swaps them. During each iteration, to increase the data reuse, the algorithm discards only one element that effectively reduces the cache capacity misses. The canneal application is included in the PARSEC benchmark to represent the engineering workloads for fine-grained parallelism and lock-free synchronization.

Dedup: The dedup application compresses the data stream. It uses the mix of global and local compression to achieve a better compression ratio. Such kind of compression is also called as deduplication. The reason to include dedup workload in the PARSEC is due to deduplication as it is the mainstream method to calculate storage footprint for the next-generation computing system. Furthermore, the dedup application is also used to compress the communication data for future generation network systems.

Streamcluster: The streamcluster application is used to solve the online clustering problem. The workload finds the median for the streams of input points and forms different clusters. Afterwards, each point is assigned to the nearest center of the cluster. It uses the sum of squared distance metric to measure the effectiveness of clustering. The reason behind the inclusion of this application in the PARSEC benchmark suite is due to increasing importance of data mining algorithms and the predominant problem of streaming characteristics.

X264: The X264 application is an H.264/AVC (Advanced Video Coding) video encoder that adds the new features in encoding such as variable block-size motion compensation (VBSMC) or context-adaptive binary arithmetic coding (CBAC), high-resolution color information, increased sample bit depth precision etc. The X264 workload allows the H.264 encoder to generate high-quality encoding output with a lower bit rate at the cost of increased encoding and decoding time. Moreover, the application uses motion compensation technique to remove the data redundancy. The X264 application is flexible and is used for fulfilling different demands such as video conferencing and HD movie distribution.

A.2.2 SPEC CPU 2006

Standard Performance Evaluation Corporation (SPEC) CPU 2006 [7], is an industry-standardized, CPU intensive benchmark suite. The SPEC benchmarks are developed to emphasize the performance of the compiler, the computer processor (CPU) and the memory architecture. It includes two benchmark suites that concentrate on two different types of compute-intensive performance.

- **CINT2006 benchmark suite:** Measures the compute-intensive integer performance. The suite contains 12 different benchmarks. The description of these workloads is given in table A.3 [7].
- **CFP2006 benchmark suite:** Measures the compute-intensive floating-point performance. The suite contains 17 different benchmarks test. The detail description of these workloads is given in table A.4 [7].

SPEC CPU 2006 suite is designed to measure the compute-intensive performance of the next generation hardware by using the programs from the real-world applications. SPEC CPU 2006 workload has different ways to quantify computer performance. One way (SPECrate metric) is to measure the number of tasks that the computer can complete in a definite time; also called a rate measurement, throughput, or capacity. Another way (SPECspeed metric) is to measure the speed with which a computer completes a single task.

Workload	Programming Language	Application Domain
400.perlbench	C	Programming Language
401.bzip2	C	Compression
403.gcc	C	C Compiler
429.mcf	C	Combinatorial Optimization
445.gobmk	C	Artificial Intelligence: Go
456.hmmer	C	Search Gene Sequence
458.sjeng	C	Artificial Intelligence: chess
462.libquantum	C	Physics / Quantum Computing
464.h264ref	C	Video Compression
471.omnetpp	C++	Discrete Event Simulation
473.astar	C++	Path-finding Algorithms
483.xalancbmk	C++	XML Processing

TABLE A.3: The inherent key characteristics of CINT2006 benchmark suite

Workload	Programming Language	Application Domain
410.bwaves	Fortran	Fluid Dynamics
416.gamess	Fortran	Quantum Chemistry
433.milc	C	Physics/Quantum Chromodynamics
434.zeusmp	Fortran	Physics / CFD
435.gromacs	C, Fortran	Biochemistry / Molecular Dynamics
436.cactusADM	C, Fortran	Physics / General Relativity
437.leslie3d	Fortran	Fluid Dynamics
444.namd	C++	Biology / Molecular Dynamics
447.dealII	C++	Finite Element Analysis
450.soplex	C++	Linear Programming, Optimization
453.povray	C++	Image Ray-tracing
454.calculix	C, Fortran	Structural Mechanics
459.GemsFDTD	Fortran	Computational Electromagnetics
465.tonto	Fortran	Quantum Chemistry
470.lbm	C	Fluid Dynamics
481.wrf	C, Fortran	Weather
482.sphinx3	C	Speech recognition

TABLE A.4: The inherent key characteristics of CFP2006 benchmark suite

A.2.2.1 Benchmark Descriptions

Here we describe the properties of some of the SPEC benchmarks that have been used in our work for the evaluation of different CMP based architectures. The detailed description about the rest of the benchmarks is reported in [7].

- **CINT2006 benchmarks**

1. **400.perlbench:** The workload is a partial version of Perl v5.8.7. It includes the email indexers: SpamAssassin and MHonArc and the tool specdiff that checks the benchmark output.
2. **401.bzip2:** The application is based on julian Seward's bzip2 version 1.0.3. All the compression and decompression process in this benchmark is done entirely in memory, rather than I/O.
3. **403.gcc:** The workload is based upon GCC ver 3.2. The workload runs as a compiler with many optimization flags enabled. It generates machine code for the AMD Opteron processor.
4. **429.mcf:** The benchmark is derived from MCF, a program used for vehicle scheduling in public mass transportation. It exploits a simple network algorithm to schedule public transport.
5. **445.gobmk:** The program plays an artificial game: Go, a simple-looking but deep complex inside.
6. **456.hmmer:** The workload is used in computational biology to search DNA sequence pattern. The application uses statistical hidden Markov model of multiple sequence alignment.
7. **458.sjeng:** The workload is based on the program Sjeng ver. 11.2 that plays chess and a variety of chess variants like losing chess and drop-chess.
8. **462.libquantum:** The workload models a quantum computer that is based on quantum mechanics and solves real hard tasks in polynomial time. To facilitate this process, it uses Shor's polynomial-time factorization algorithm.

9. **464.h264ref:** The workload is an implementation of H.264/AVC coding technique that is expected to replace MPEG2.
10. **471.omnetpp:** The workload models a vast ethernet network using discrete event simulation.
11. **473.astar:** The workload is derived from the well known 2D-path finding libraries used in AI games. It models different variants of A* path-finding algorithms based upon the requirement.

- **CFP2006 benchmarks**

1. **410.bwaves:** The workload models the blast wave as a three dimensional transonic transient laminar viscous flow.
2. **416.gamess:** The workload models the different varieties of quantum chemical computations. It performs the self-consistent field calculations using Multi-Configuration Self-Consistent Field, Restricted Hartree Fock method, and Restricted open-shell Hartree-Fock.
3. **433.milc:** The workload models the four-dimensional SU(3) lattice gauge theory using Von-Neumann MIMD parallel machines.
4. **434.zeusmp:** The workload models the astrophysical phenomena. The application resolves the problems in three spatial dimensions with a wide variety of boundary constraints.
5. **435.gromacs:** It is used to perform molecular dynamics. It models the Newtonian equations of motion for systems with hundreds to millions of particles.
6. **436.cactusADM:** The workload is a combination of Cactus, an open-source problem-solving environment, and BenchADM, kernel representative of numerical relativity. The application solves the Einstein evolution equation using leapfrog numerical method.
7. **437.leslie3d:** The workload is based on LESlie3d (Large-Eddy Simulations with Linear-Eddy Model in 3D). It solves the problem of Computational Fluid Dynamics (CFD) using MacCormack predictor-corrector time integration scheme.

8. **444.namd:** The workload models the large bio-molecular systems. It tests the atoms of apolipoprotein A-I.
9. **447.dealII:** The workload is based on deal.II, a library targeted at adaptive finite elements and error estimation. The application provides a solution for the Helmholtz-type equation with non-constant coefficients.
10. **450.soplex:** The application provides solution for the linear program using a simplex algorithm and sparse linear algebra.
11. **454.calculix:** The workload is derived from CalculiX, finite element code for linear and nonlinear three-dimensional structural application. The application provides the solution for buckling, eigen mode analysis, etc.
12. **465.tonto:** The workload is an open-source quantum chemistry package. It performs the calculation of Hartree-Fock wave function to match experimental X-ray diffraction data.
13. **470.lbm:** The workload implements Lattice-Boltzmann Method to model incompressible fluid in 3D.

A.3 Simulation Procedure

In this dissertation, we have used several multi-threaded and multi-programmed benchmarks for the simulation analysis. This section illustrates all the multi-programmed and multi-threaded benchmarks that we made from the SPEC CPU 2006 and PARSEC benchmarks.

A.3.1 Multi-threaded vs Multi-programmed Workloads

Every benchmark in the PARSEC suite is the multi-threaded workload. The number of threads in each program depends upon the input size and load of the program. Most of the benchmarks take the number of threads as a command-line argument. In all PARSEC benchmarks, during the execution, the multi-threading

Multi-threaded Benchmarks			
Bodytrack, Canneal, Dedup, Fluidanimate, Freqmine, Streamcluster, Swaptions, X264			
Multi-programmed Benchmarks			
Dual Core		**Quad Core**	
Mixes	**Details**	**Mixes**	**Details**
Mix1	lbm, cactusADM	Mix1	lbm, mcf, bwaves, leslie3d
Mix2	mcf, zeusmp	Mix2	zeusmp, gromacs, gamess, cactusADM
Mix3	leslie3d, gromacs	Mix3	lbm, mcf, gromacs, gamess
Mix4	leslie3d, gamess	Mix4	perlbench, gcc, omnetpp, libquantum
Mix5	perlbench, bzip2	Mix5	milc, hmmer, bzip2, soplex
Mix6	mcf, cactusADM	Mix6	perlbench, gcc, milc, hmmer
Mix7	omnetpp, hmmer	Mix7	dealII, sjeng, h264ref, tonto
Mix8	gcc, bzip2	Mix8	gobmk, calculix, astar, namd
Mix9	dealII, namd	Mix9	gobmk, tonto, sjeng, namd
Mix10	calculix, tonto	-	-
Mix11	h264ref, gobmk	-	-
Mix12	dealII, gobmk	-	-

TABLE A.5: List of all the multi-threaded and multi-programmed benchmarks used for the simulations in this dissertation

occurs in a specific period called Region Of Interest (ROI). In particular, the real PARSEC application execution happens in the ROI. The input scanning, initialization of the variable, etc. are performed before ROI, and once the ROI execution is completed, the workload terminates after generating the output.

The multi-programmed benchmarks are built by merging multiple SPEC CPU 2006 benchmarks through m5 commands in the virtual target system. The phrase 'merging' implies that different processes are executing on the different cores until a process completes the specified number of instructions. For instance, by running the 4 SPEC process like bzip2, mcf, milc, and leslie3d, we can make a multi-programmed mix for a four-core CMP. In multi-programmed benchmarks, the phrase benchmark represents the combined workloads.

A.3.2 Benchmarks Used in Our Simulations

Based on the multi-threaded and multi-programmed workloads provided by PARSEC and SPEC CPU 2006, different mixtures of the workloads for the simulation can be made. These mixtures can be either of single PARSEC benchmark application with multiple threads or a mix of different processes from the SPEC CPU 2006. Table A.5 presents the details of the benchmarks that we used in our simulation analysis.

A.3.3 Benchmark Running Process

To run multi-threaded benchmarks, we run a PARSEC workload on the target machine up to the completion of the workload. In this process, four stats are dumped in the generated stats file that comprises of (a) statistics for M5 full system booting process, (b) statistic before reaching ROI that include the initialization of benchmarks and the spawning of the threads, (c) the statistics in the ROI and (d) the statistics from the end of the ROI to the simulation exit. Our focus is on the third stats that is ROI which represents the actual execution of the PARSEC applications.

On the other hand, to run multi-programmed benchmarks, the very first step is to load all the applications one by one. Each application is then executed for 250 million instructions for warm-up. Here, the warm-up phase is essential to go beyond the compulsory misses in the cache, which allows the proposed architecture to settle properly in the simulator. After warming-up, each workload is run for one billion instructions to collect the required stats needed to analyze the performance of the proposed design.

A.3.4 Comparing Different CMP Architecture

The effectiveness of the proposed CMP architecture with other existing architectures are compared in terms of IPC, energy consumption, EDP, lifetime, write

variation, implementation overhead, etc. To facilitate this process, we have implemented all our proposed and prior designs on GEM5 (full system simulator). On top of the framework, we execute different PARSEC and SPEC CPU 2006 workloads. Different variety of statistics are recorded during the execution of each workload as reported in section A.1.1.5. Based on the stats generated, the efficacy of architectures is evaluated.

Generally, the architecture is engineered with different choices and design configurations, for instance, with different cache associativity, various cache sizes, block sharing capabilities, etc. Whenever needed, we will provide the appropriate details in the relevant chapters/sections. For all the architectures with a different configuration, the process of executing a benchmark is kept the same to maintain regularity. Separate result of each workload is illustrated, and the geometric mean of all benchmarks are derived in our result sections.

Bibliography

[1] G. E. Moore, "Cramming More Components Onto Integrated Circuits," *Proceedings of the IEEE*, vol. 86, no. 1, pp. 82–85, Jan 1998.

[2] P. Kongetira, K. Aingaran, and K. Olukotun, "Niagara: A 32-Way Multithreaded Sparc Processor," *IEEE Micro*, vol. 25, no. 2, pp. 21–29, March 2005.

[3] [Online]. Available: http://www.intel.com

[4] [Online]. Available: https://www.oracle.com/sun/index.html

[5] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Fourth Edition: A Quantitative Approach.* Morgan Kaufmann Publishers Inc., 2006.

[6] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '08, 2008, pp. 72–81.

[7] J. L. Henning, "SPEC CPU2006 Benchmark Descriptions," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, sept 2006.

[8] D. E. Culler and J. P. Singh, *Parallel Computer Architecture: A Hardware/-Software Approach.* Morgan Kaufmann Publishers, 1999.

[9] C. Bienia, S. Kumar, and K. Li, "PARSEC vs. SPLASH-2: A quantitative comparison of two multithreaded benchmark suites on Chip-Multiprocessors," in *IEEE International Symposium on Workload Characterization (IISWC)*, sept 2008, pp. 47–56.

[10] "International Technology Roadmap for Semiconductors - ITRS 2.0," http://www.itrs2.net/, [Online].

[11] R. Balasubramonian, N. P. Jouppi, and N. Muralimanohar, "Multi-Core Cache hierarchies," *Synthesis Lectures on Computer Architecture*, vol. 6, no. 3, pp. 1–153, 2011.

[12] W. Zang and A. Gordon-Ross, "A Survey on Cache Tuning from a Power/Energy Perspective," *ACM Comput. Surv.*, vol. 45, no. 3, pp. 32:1–32:49, Jul. 2013.

[13] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures," in *42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec 2009, pp. 469–480.

[14] S. Kaxiras and M. Martonosi, *COMPUTER ARCHITECTURE TECHNIQUES FOR POWER-EFFICIENCY*. Mark D. Hill, University of Wisconsin, Madison, 2008.

[15] A. P. Chandrakasan and R. W. Brodersen, *Low Power Digital CMOS Design*. Norwell, MA, USA: Kluwer Academic Publishers, 1995.

[16] K. Roy and S. C. Prasad, *Low-Power CMOS VLSI Circuit Design*. John Wiley and Sons, 2009.

[17] Y.-C. Yeo, T.-J. King, and C. Hu, "MOSFET Gate Leakage Modeling and Selection Guide for Alternative Gate Dielectrics based on Leakage Considerations," *IEEE Transactions on Electron Devices*, vol. 50, no. 4, pp. 1027–1035, April 2003.

[18] V. Hanumaiah, S. Vrudhula, and K. S. Chatha, "Maximizing Performance of Thermally Constrained Multi-Core Processors by Dynamic Voltage and Frequency Control," in *IEEE/ACM International Conference on Computer-Aided Design - Digest of Technical Papers*, Nov 2009, pp. 310–313.

[19] R. B. Naveen Muralimanohar and N. P. Jouppi, "CACTI 6.0: A Tool to Model Large Caches," in *Technical Report HPL-2009-85, HP Laboratories*, 2007.

[20] S. Mittal, J. S. Vetter, and D. Li, "A Survey Of Architectural Approaches for Managing Embedded DRAM and Non-Volatile On-Chip Caches," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 6, pp. 1524–1537, June 2015.

[21] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory," *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, vol. 31, no. 7, pp. 994–1007, 2012.

[22] M. Poremba, S. Mittal, D. Li, J. S. Vetter, and Y. Xie, "DESTINY: A Tool for Modeling Emerging 3D NVM and eDRAM Caches," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2015, pp. 1543–1546.

[23] H.-S. Wang, X. Zhu, L.-S. Peh, and S. Malik, "ORION: A Power-Performance Simulator for Interconnection Networks," in *35th Annual IEEE/ACM International Symposium on Microarchitecture, 2002. (MICRO-35). Proceedings.*, 2002, pp. 294–305.

[24] S. Mittal, "A survey of architectural techniques for improving cache power efficiency," *Sustainable Computing: Informatics and Systems*, vol. 4, no. 1, pp. 33 – 43, 2014.

[25] B. M. Rogers, A. Krishna, G. B. Bell, K. Vu, X. Jiang, and Y. Solihin, "Scaling the Bandwidth Wall: Challenges in and Avenues for CMP Scaling," *SIGARCH Comput. Archit. News*, vol. 37, no. 3, pp. 371–382, Jun. 2009. [Online]. Available: http://doi.acm.org/10.1145/1555815.1555801

[26] J. Boukhobza, S. Rubini, R. Chen, and Z. Shao, "Emerging NVM: A Survey on Architectural Integration and Research Challenges," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 23, no. 2, pp. 14:1–14:32, Nov. 2017.

[27] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, and Y. Xie, "Hybrid Cache Architecture with Disparate Memory Technologies," in *ACM SIGARCH computer architecture news*, vol. 37, no. 3. ACM, 2009, pp. 34–45.

[28] X. Wu, J. Li, L. Zhang, E. Speight, and Y. Xie, "Power and Performance of Read-Write aware Hybrid Caches with Non-Volatile Memories," in *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2009, pp. 737–742.

[29] J. Wang, X. Dong, Y. Xie, and N. P. Jouppi, "i2WAP: Improving Non-Volatile Cache Lifetime by reducing Inter- and Intra-set Write Variations," in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2013, pp. 234–245.

[30] J. Ahn, S. Yoo, and K. Choi, "Write Intensity Prediction for Energy-Efficient Non-Volatile Caches," in *International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2013, pp. 223–228.

[31] S. Mittal and J. S. Vetter, "EqualChance: Addressing Intra-set Write Variation to Increase Lifetime of Non-volatile Caches," in *2nd Workshop on Interactions of NVM/Flash with Operating Systems and Workloads (INFLOW 14)*. Broomfield, CO: USENIX Association, 2014. [Online]. Available: https://www.usenix.org/conference/inflow14/workshop-program/presentation/mittal

[32] M. R. Jokar, M. Arjomand, and H. Sarbazi-Azad, "Sequoia: A High-Endurance NVM-Based Cache Architecture," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 3, pp. 954–967, March 2016.

[33] S. Mittal, J. S. Vetter, and D. Li, "WriteSmoothing: Improving Lifetime of Non-volatile Caches Using Intra-set Wear-leveling," in *Proceedings of the*

24th Edition of the Great Lakes Symposium on VLSI, ser. GLSVLSI '14. New York, NY, USA: ACM, 2014, pp. 139–144.

[34] N. Gulur, M. Mehendale, R. Manikantan, and R. Govindarajan, "Bi-Modal DRAM Cache: Improving Hit Rate, Hit Latency and Bandwidth," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec 2014, pp. 38–50.

[35] Z. Jaki and R. Canal, "DRAM-based Coherent Caches and How to take Advantage of the Coherence Protocol to reduce the Refresh Energy," in *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2014, pp. 1–4.

[36] H. Jang, Y. Lee, J. Kim, Y. Kim, J. Kim, J. Jeong, and J. W. Lee, "Efficient Footprint Caching for Tagless DRAM Caches," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, March 2016, pp. 237–248.

[37] M. H. Kryder and C. S. Kim, "After hard drivesWhat comes next?" *IEEE Transactions on Magnetics*, vol. 45, no. 10, pp. 3406–3413, 2009.

[38] D. Apalkov, A. Khvalkovskiy, S. Watts, V. Nikitin, X. Tang, D. Lottis, K. Moon, X. Luo, E. Chen, A. Ong, A. Driskill-Smith, and M. Krounbi, "Spin-transfer Torque Magnetic Random Access Memory (STT-MRAM)," *J. Emerg. Technol. Comput. Syst.*, vol. 9, no. 2, pp. 13:1–13:35, May 2013.

[39] Y. Huai *et al.*, "Spin-transfer torque MRAM (STT-MRAM): Challenges and prospects," *AAPPS bulletin*, vol. 18, no. 6, pp. 33–40, 2008.

[40] H. Li and Y. Chen, "An Overview of Non-Volatile Memory Technology and the Implication for Tools and Architectures," in *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2009, pp. 731–736.

[41] X. Dong, X. Wu, G. Sun, Y. Xie, H. Li, and Y. Chen, "Circuit and Microarchitecture Evaluation of 3D Stacking Magnetic RAM (MRAM) as a Universal Memory Replacement," in *2008 45th ACM/IEEE Design Automation Conference*. IEEE, 2008, pp. 554–559.

[42] S. . Chung, T. Kishi, J. W. Park, M. Yoshikawa, K. S. Park, T. Nagase, K. Sunouchi, H. Kanaya, G. C. Kim, K. Noma, M. S. Lee, A. Yamamoto, K. M. Rho, K. Tsuchida, S. J. Chung, J. Y. Yi, H. S. Kim, Y. S. Chun, H. Oyamatsu, and S. J. Hong, "4Gbit Density STT-MRAM using Perpendicular MTJ realized with Compact Cell Structure," in *2016 IEEE International Electron Devices Meeting (IEDM)*, Dec 2016, pp. 27.1.1–27.1.4.

[43] Y. Lu, T. Zhong, W. Hsu, S. Kim, X. Lu, J. J. Kan, C. Park, W. C. Chen, X. Li, X. Zhu, P. Wang, M. Gottwald, J. Fatehi, L. Seward, J. P. Kim, N. Yu, G. Jan, J. Haq, S. Le, Y. J. Wang, L. Thomas, J. Zhu, H. Liu, Y. J. Lee, R. Y. Tong, K. Pi, D. Shen, R. He, Z. Teng, V. Lam, R. Annapragada, T. Torng, P. Wang, and S. H. Kang, "Fully functional perpendicular stt-mram macro embedded in 40 nm logic for energy-efficient iot applications," in *2015 IEEE International Electron Devices Meeting (IEDM)*, Dec 2015, pp. 26.1.1–26.1.4.

[44] O. Golonzka, J. . Alzate, U. Arslan, M. Bohr, P. Bai, J. Brockman, B. Buford, C. Connor, N. Das, B. Doyle, T. Ghani, F. Hamzaoglu, P. Heil, P. Hentges, R. Jahan, D. Kencke, B. Lin, M. Lu, M. Mainuddin, M. Meterelliyoz, P. Nguyen, D. Nikonov, K. O'brien, J. O. Donnell, K. Oguz, D. Ouellette, J. Park, J. Pellegren, C. Puls, P. Quintero, T. Rahman, A. Romang, M. Sekhar, A. Selarka, M. Seth, A. J. Smith, A. K. Smith, L. Wei, C. Wiegand, Z. Zhang, and K. Fischer, "MRAM as Embedded Non-Volatile Memory Solution for 22FFL FinFET Technology," in *2018 IEEE International Electron Devices Meeting (IEDM)*, Dec 2018, pp. 18.1.1–18.1.4.

[45] Y. J. Song, J. H. Lee, S. H. Han, H. C. Shin, K. H. Lee, K. Suh, D. E. Jeong, G. H. Koh, S. C. Oh, J. H. Park, S. O. Park, B. J. Bae, O. I. Kwon, K. H. Hwang, B. Y. Seo, Y. K. Lee, S. H. Hwang, D. S. Lee, Y. Ji, K. C.

Park, G. T. Jeong, H. S. Hong, K. P. Lee, H. K. Kang, and E. S. Jung, "Demonstration of Highly Manufacturable STT-MRAM Embedded in 28nm Logic," in *2018 IEEE International Electron Devices Meeting (IEDM)*, Dec 2018, pp. 18.2.1–18.2.4.

[46] S. Raoux, G. W. Burr, M. J. Breitwisch, C. T. Rettner, Y. . Chen, R. M. Shelby, M. Salinga, D. Krebs, S. . Chen, H. . Lung, and C. H. Lam, "Phase-Change Random Access Memory: A Scalable Technology," *IBM Journal of Research and Development*, vol. 52, no. 4.5, pp. 465–479, July 2008.

[47] G. W. Burr, B. N. Kurdi, J. C. Scott, C. H. Lam, K. Gopalakrishnan, and R. S. Shenoy, "Overview of Candidate Device Technologies for Storage-Class Memory," *IBM Journal of Research and Development*, vol. 52, no. 4.5, pp. 449–464, July 2008.

[48] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting Phase Change Memory as a Scalable DRAM Alternative," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ser. ISCA '09. New York, NY, USA: ACM, 2009, pp. 2–13.

[49] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A Durable and Energy Efficient Main Memory Using Phase Change Memory Technology," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ser. ISCA '09. New York, NY, USA: ACM, 2009, pp. 14–23.

[50] K. Lee, B. Cho, W. Cho, S. Kang, B. Choi, H. Oh, C. Lee, H. Kim, J. Park, Q. Wang, M. Park, Y. Ro, J. Choi, K. Kim, Y. Kim, I. Shin, K. Lim, H. Cho, C. Choi, W. Chung, D. Kim, K. Yu, G. Jeong, H. Jeong, C. Kwak, C. Kim, and K. Kim, "A 90nm 1.8V 512Mb Diode-Switch PRAM with 266MB/s Read Throughput," in *2007 IEEE International Solid-State Circuits Conference. Digest of Technical Papers*, Feb 2007, pp. 472–616.

[51] Y. Choi, I. Song, M. Park, H. Chung, S. Chang, B. Cho, J. Kim, Y. Oh, D. Kwon, J. Sunwoo, J. Shin, Y. Rho, C. Lee, M. G. Kang, J. Lee, Y. Kwon, S. Kim, J. Kim, Y. Lee, Q. Wang, S. Cha, S. Ahn, H. Horii, J. Lee, K. Kim,

H. Joo, K. Lee, Y. Lee, J. Yoo, and G. Jeong, "A 20nm 1.8V 8Gb PRAM with 40MB/s Program Bandwidth," in *2012 IEEE International Solid-State Circuits Conference*, Feb 2012, pp. 46–48.

[52] H. Akinaga and H. Shima, "Resistive Random Access Memory (ReRAM) Based on Metal Oxides," *Proceedings of the IEEE*, vol. 98, no. 12, pp. 2237–2251, Dec 2010.

[53] J. J. Yang, M. D. Pickett, X. Li, D. A. Ohlberg, D. R. Stewart, and R. S. Williams, "Memristive Switching Mechanism for Metal/Oxide/Metal Nanodevices," *Nature nanotechnology*, vol. 3, no. 7, p. 429, 2008.

[54] Y. Ho, G. M. Huang, and P. Li, "Non-Volatile Memristor Memory: Device Characteristics and Design Implications," in *2009 IEEE/ACM International Conference on Computer-Aided Design - Digest of Technical Papers*, Nov 2009, pp. 485–490.

[55] D. L. Lewis and H. S. Lee, "Architectural Evaluation of 3D Stacked RRAM Caches," in *2009 IEEE International Conference on 3D System Integration*, Sep. 2009, pp. 1–4.

[56] R. S. Williams, "How We Found The Missing Memristor," *IEEE Spectrum*, vol. 45, no. 12, pp. 28–35, Dec 2008.

[57] "What Happened To ReRAM?" [Online]. Available: https://semiengineering.com/what-happened-to-reram/

[58] J. Li, C. J. Xue, and Y. Xu, "STT-RAM based Energy-Efficiency Hybrid Cache for CMPs," in *2011 IEEE/IFIP 19th International Conference on VLSI and System-on-Chip*. IEEE, 2011, pp. 31–36.

[59] Y.-T. Chen, J. Cong, H. Huang, C. Liu, R. Prabhakar, and G. Reinman, "Static and Dynamic Co-Optimizations for Blocks Mapping in Hybrid Caches," in *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design*. ACM, 2012, pp. 237–242.

[60] S. Guo, Z. Liu, D. Wang, H. Wang, and G. Li, "Wear-Resistant Hybrid Cache Architecture with Phase Change Memory," in *2012 IEEE Seventh International Conference on Networking, Architecture, and Storage*. IEEE, 2012, pp. 268–272.

[61] 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 *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2014, pp. 13–24.

[62] J. Wang, Y. Tim, W.-F. Wong, Z.-L. Ong, Z. Sun, and H. H. Li, "A Coherent Hybrid SRAM and STT-RAM L1 Cache Architecture for Shared Memory Multicores," in *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2014, pp. 610–615.

[63] D. Chen, H. Jin, X. Liao, H. Liu, R. Guo, and D. Liu, "MALRU: Miss-Penalty Aware LRU-based Cache Replacement for Hybrid Memory Systems," in *Proceedings of the Conference on Design, Automation & Test in Europe*. European Design and Automation Association, 2017, pp. 1086–1091.

[64] L. Wang, F. Ge, H. Lu, N. Wu, Y. Zhang, and F. Zhou, "A Spherical Placement and Migration Scheme for a STT-RAM Based Hybrid Cache in 3D chip Multi-processors," in *Proceedings of the World Congress on Engineering*, vol. 1, 2018.

[65] B. Quan, T. Zhang, T. Chen, and J. Wu, "Prediction Table based Management Policy for STT-RAM and SRAM Hybrid Cache," in *2012 7th International Conference on Computing and Convergence Technology (ICCCT)*. IEEE, 2012, pp. 1092–1097.

[66] J. Ahn, S. Yoo, and K. Choi, "Prediction Hybrid Cache: An Energy-Efficient STT-RAM Cache Architecture," *IEEE Transactions on Computers*, vol. 65, no. 3, pp. 940–951, 2015.

[67] N. Kim, J. Ahn, W. Seo, and K. Choi, "Energy-Efficient Exclusive Last-Level Hybrid Caches consisting of SRAM and STT-RAM," in *2015 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*. IEEE, 2015, pp. 183–188.

[68] J. He and J. Callenes-Sloan, "A Novel Architecture of Large Hybrid Cache with Reduced Energy," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 12, pp. 3092–3102, 2017.

[69] ——, "Architecting a Novel Hybrid Cache with Low Energy," in *2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 2017, pp. 370–370.

[70] J. Wang, X. Dong, and Y. Xie, "OAP: An Obstruction-Aware Cache Management Policy for STT-RAM Last-Level Caches," in *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 2013, pp. 847–852.

[71] J. Ahn, S. Yoo, and K. Choi, "DASCA: Dead write prediction Assisted STT-RAM Cache Architecture," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2014, pp. 25–36.

[72] C. Zhang, G. Sun, P. Li, T. Wang, D. Niu, and Y. Chen, "SBAC: A Statistics based cache Bypassing method for Asymmetric-access Caches," in *Proceedings of the 2014 international symposium on Low power electronics and design*. ACM, 2014, pp. 345–350.

[73] G. Sun, C. Zhang, P. Li, T. Wang, and Y. Chen, "Statistical Cache Bypassing for Non-Volatile Memory," *IEEE Transactions on Computers*, vol. 65, no. 11, pp. 3427–3440, 2016.

[74] H. Kim, S. Kim, and J. Lee, "Write-Amount-Aware Management Policies for STT-RAM Caches," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 4, pp. 1588–1592, 2016.

[75] K. Korgaonkar, I. Bhati, H. Liu, J. Gaur, S. Manipatruni, S. Subramoney, T. Karnik, S. Swanson, I. Young, and H. Wang, "Density Tradeoffs of Non-Volatile Memory as a Replacement for SRAM based Last Level Cache," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*. IEEE Press, 2018, pp. 315–327.

[76] D. Lee and K. Choi, "Energy-Efficient Partitioning of Hybrid Caches in Multi-Core Architecture," in *IFIP/IEEE International Conference on Very Large Scale Integration-System on a Chip*. Springer, 2014, pp. 58–74.

[77] C. Lin and J.-N. Chiou, "High-Endurance Hybrid Cache Design in CMP Architecture with Cache Partitioning and Access-Aware Policies," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 10, pp. 2149–2161, 2014.

[78] W. Wei, D. Jiang, J. Xiong, and M. Chen, "HAP: Hybrid-Memory-Aware Partition in Shared Last-Level Cache," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 14, no. 3, p. 24, 2017.

[79] M. Bakhshalipour, A. Faraji, S. A. V. Ghahani, F. Samandi, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Reducing Writebacks through In-Cache Displacement," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 24, no. 2, p. 16, 2019.

[80] R. Pagh and F. F. Rodler, "Cuckoo Hashing," *Journal of Algorithms*, vol. 51, no. 2, pp. 122–144, 2004.

[81] N. El-Sayed, A. Mukkara, P.-A. Tsai, H. Kasture, X. Ma, and D. Sanchez, "KPart: A Hybrid Cache Partitioning-Sharing Technique for Commodity Multicores," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2018, pp. 104–117.

[82] Z. Sun, X. Bi, H. Li, W. Wong, Z. Ong, X. Zhu, and W. Wu, "Multi Retention Level STT-RAM Cache Designs with a Dynamic Refresh Scheme," in *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec 2011, pp. 329–338.

[83] C. W. Smullen, V. Mohan, A. Nigam, S. Gurumurthi, and M. R. Stan, "Relaxing Non-Volatility for fast and Energy-Efficient STT-RAM Caches," in *2011 IEEE 17th International Symposium on High Performance Computer Architecture*, Feb 2011, pp. 50–61.

[84] 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 *DAC Design Automation Conference 2012*, June 2012, pp. 243–252.

[85] Q. Li, Y. He, J. Li, L. Shi, Y. Chen, and C. J. Xue, "Compiler-Assisted Refresh Minimization for Volatile STT-RAM Cache," *IEEE Transactions on Computers*, vol. 64, no. 8, pp. 2169–2181, Aug 2015.

[86] M. Wang, Z. Sun, J. Diao, X. Wang, N. Li, and K. Bu, "A Study on Reconfiguring On-Chip Cache with Non-Volatile Memory," in *2014 11th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, May 2014, pp. 97–99.

[87] B. Yang, J. Lee, J. Kim, J. Cho, S. Lee, and B. Yu, "A Low Power Phase-Change Random Access Memory using a Data-Comparison Write Scheme," in *2007 IEEE International Symposium on Circuits and Systems*, May 2007, pp. 3014–3017.

[88] S. Cho and H. Lee, "Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance," in *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec 2009, pp. 347–357.

[89] S. Niknam, A. Asad, M. Fathy, and A. Rahmani, "Energy Efficient 3D Hybrid Processor-Memory Architecture for the Dark Silicon Age," in *2015 10th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, June 2015, pp. 1–8.

[90] Y. Chen, J. Cong, H. Huang, B. Liu, C. Liu, M. Potkonjak, and G. Reinman, "Dynamically Reconfigurable Hybrid Cache: An Energy-Efficient Last-Level

Cache Design," in *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2012, pp. 45–50.

[91] T. Adegbija, "Exploring Configurable Non-Volatile Memory-based Caches for Energy-Efficient Embedded Systems," in *2016 International Great Lakes Symposium on VLSI (GLSVLSI)*, May 2016, pp. 157–162.

[92] K. Kuan and T. Adegbija, "Energy-efficient runtime adaptable l1 stt-ram cache design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2019.

[93] ——, "Halls: An energy-efficient highly adaptable last level stt-ram cache for multicore systems," *IEEE Transactions on Computers*, pp. 1–1, 2019.

[94] S. Mittal, J. S. Vetter, and D. Li, "LastingNVCache: A Technique for Improving the Lifetime of Non-volatile Caches," in *2014 IEEE Computer Society Annual Symposium on VLSI*, July 2014, pp. 534–540.

[95] S. Mittal and J. Vetter, "A Technique for Improving Lifetime of Non-Volatile Caches Using Write-Minimization," *Journal of Low Power Electronics and Applications*, vol. 6, no. 1, 2016. [Online]. Available: https://www.mdpi.com/2079-9268/6/1/1

[96] S. Mittal and J. S. Vetter, "EqualWrites: Reducing Intra-Set Write Variations for Enhancing Lifetime of Non-Volatile Caches," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 1, pp. 103–114, Jan 2016.

[97] E. Reed, A. R. Alameldeen, H. Naeimi, and P. Stolt, "Probabilistic Replacement Strategies for Improving the Lifetimes of NVM-based Caches," in *Proceedings of the International Symposium on Memory Systems*, ser. MEMSYS '17. New York, NY, USA: ACM, 2017, pp. 166–176.

[98] 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," *J. Emerg. Technol. Comput. Syst.*, vol. 9, no. 2, pp. 16:1–16:22, May 2013.

[99] S. Mittal, "Using Cache-Coloring to Mitigate Inter-set Write Variation in Non-volatile Caches," *CoRR*, vol. abs/1310.8494, 2013. [Online]. Available: http://arxiv.org/abs/1310.8494

[100] S. Mittal and J. S. Vetter, "Addressing Inter-Set Write-Variation for improving Lifetime of Non-Volatile Caches," in *5th Annual Non-Volatile Memories Workshop*. University of California, San Diego, 2014.

[101] M. Soltani, M. Ebrahimi, and Z. Navabi, "Prolonging Lifetime of Non-Volatile Last Level Caches with Cluster Mapping," in *2016 International Great Lakes Symposium on VLSI (GLSVLSI)*, May 2016, pp. 329–334.

[102] Y. Joo, D. Niu, X. Dong, G. Sun, N. Chang, and Y. Xie, "Energy- and Endurance-Aware Design of Phase Change Memory Caches," in *2010 Design, Automation Test in Europe Conference Exhibition (DATE 2010)*, March 2010, pp. 136–141.

[103] S. Yazdanshenas, M. R. Pirbasti, M. Fazeli, and A. Patooghy, "Coding Last Level STT-RAM Cache for High Endurance and Low Power," *IEEE Computer Architecture Letters*, vol. 13, no. 2, pp. 73–76, July 2014.

[104] S. Asadi, A. M. H. Monazzah, H. Farbeh, and S. G. Miremadi, "WIPE: Wearout Informed Pattern Elimination to Improve the Endurance of NVM-based Caches," in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan 2017, pp. 188–193.

[105] S. Wang, G. Duan, Y. Li, and Q. Dong, "Word- and Partition-Level Write Variation Reduction for Improving Non-Volatile Cache Lifetime," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 23, no. 1, pp. 4:1–4:18, Aug. 2017.

[106] M. K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali, "Enhancing Lifetime and Security of PCM-based Main Memory

with Start-Gap Wear Leveling," in *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec 2009, pp. 14–23.

[107] S. Mittal and J. S. Vetter, "AYUSH: Extending Lifetime of SRAM-NVM Way-Based Hybrid Caches Using Wear-Leveling," in *2015 IEEE 23rd International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, Oct 2015, pp. 112–121.

[108] D. B. Strukov, "Endurance-Write-Speed Tradeoffs in Non-Volatile Memories," *Applied Physics A*, vol. 122, no. 4, p. 302, 2016.

[109] J. B. Kotra, M. Arjomand, D. Guttman, M. T. Kandemir, and C. R. Das, "Re-nuca: A practical nuca architecture for reram based last-level caches," in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2016, pp. 576–585.

[110] H. Farbeh, A. M. H. Monazzah, E. Aliagha, and E. Cheshmikhani, "A-cache: Alternating cache allocation to conduct higher endurance in nvm-based caches," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 66, no. 7, pp. 1237–1241, July 2019.

[111] S. A. Wolf, J. Lu, M. R. Stan, E. Chen, and D. M. Treger, "The Promise of Nanomagnetics and Spintronics for Future Logic and Universal Memory," *Proceedings of the IEEE*, vol. 98, no. 12, pp. 2155–2168, 2010.

[112] M. Lodde, J. Flich, and M. E. Acacio, "Dynamic last-level cache allocation to reduce area and power overhead in directory coherence protocols," in *Euro-Par 2012 Parallel Processing*, C. Kaklamanis, T. Papatheodorou, and P. G. Spirakis, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 206–218.

[113] J. Albericio, P. Ibez, V. Vials, and J. M. Llabera, "The reuse cache: Downsizing the shared last-level cache," in *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec 2013, pp. 310–321.

[114] J. L. Carter and M. N. Wegman, "Universal Classes of Hash Functions (Extended Abstract)," in *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, ser. STOC '77. New York, NY, USA: ACM, 1977, pp. 106–112.

[115] P. Pujara and A. Aggarwal, "Cache Noise Prediction," *IEEE Transactions on Computers*, vol. 57, no. 10, pp. 1372–1386, Oct 2008.

[116] C. F. Chen et al., "Accurate and Complexity-Effective Spatial Pattern Prediction," in *10th International Symposium on High Performance Computer Architecture (HPCA'04)*. IEEE, 2004, pp. 276–287.

[117] M. A. Alves, C. Villavieja, M. Diener, and P. O. Navaux, "Energy Efficient Last Level Caches via Last Read/Write Prediction," in *2013 25th International Symposium on Computer Architecture and High Performance Computing*, Oct 2013, pp. 73–80.

[118] 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," *ACM SIGARCH Computer Architure News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[119] A. Jaleel, "Memory Characterization of Workloads using Instrumentation-Driven Simulation a Pin-based Memory Characterization of the SPEC CPU2000 and SPEC CPU2006 Benchmark Suites," VSSAD, Tech. Rep., 2007.

[120] J. Liao, F. Zhang, L. Li, and G. Xiao, "Adaptive wear-leveling in flash-based memory," *IEEE Computer Architecture Letters*, vol. 14, no. 1, pp. 1–4, Jan 2015.

[121] M. Hosomi, H. Yamagishi, T. Yamamoto, K. Bessho, Y. Higo, K. Yamane, H. Yamada, M. Shoji, H. Hachino, C. Fukumoto, H. Nagao, and H. Kano, "A Novel Non-Volatile Memory with Spin Torque Transfer Magnetization

Switching: Spin-RAM," in *IEEE InternationalElectron Devices Meeting, 2005. IEDM Technical Digest.*, Dec 2005, pp. 459–462.

[122] N. P. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers," in *[1990] Proceedings. The 17th Annual International Symposium on Computer Architecture*, May 1990.

[123] Q. Wang, J.-R. Li, and D.-H. Wang, "Improving the Performance and Energy Efficiency of Phase Change Memory Systems," *Journal of Computer Science and Technology*, vol. 30, no. 1, pp. 110–120, Jun. 2015. [Online]. Available: https://doi.org/10.1007/s11390-015-1508-3

[124] B. Pourshirazi, M. V. Beigi, Z. Zhu, and G. Memik, "Writeback-aware llc management for pcm-based main memory systems," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 24, no. 2, Jan. 2019. [Online]. Available: https://doi.org/10.1145/3292009

[125] R. Jain, "The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling," *SIGMETRICS Performance Evaluation Review*, vol. 19, pp. 5–11, 1991.

[126] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A Full System Simulation Platform," *Computer*, vol. 35, no. 2, pp. 50–58, Feb 2002.

[127] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's general execution-driven multiprocessor simulator (gems) toolset," *SIGARCH Comput. Archit. News*, vol. 33, no. 4, pp. 92–99, 2005.

[128] N. Binkert, R. Dreslinski, L. Hsu, K. Lim, A. Saidi, and S. Reinhardt, "The M5 Simulator: Modeling Networked Systems," *IEEE Micro*, vol. 26, no. 4, pp. 52–60, Jul. 2006.

[129] M. Rosenblum and M. Varadarajan, "SimOS: A Fast Operating System Simulation Environment," Stanford, CA, USA, Tech. Rep., 1994.

[130] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: an infrastructure for computer system modeling," *Computer*, vol. 35, no. 2, pp. 59–67, Feb. 2002.

[131] S. K. Sastry Hari, M.-L. Li, P. Ramachandran, B. Choi, and S. V. Adve, "mSWAT: Low-cost Hardware Fault Detection and Diagnosis for Multicore Systems," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2009, pp. 122–132.

[132] L. Schaelicke and M. Parker, "The Design and Utility of the ML-RSIM System Simulator," *Journal of Systems Architecture*, vol. 52, no. 5, pp. 283–297, May. 2006.

[133] J. An, X. Fan, S. Zhang, and D. Wang, "An Efficient Verification Method for Microprocessors Based on the Virtual Machine," in *Proceedings of the 1st International Conference on Embedded Software and Systems*, 2005, pp. 514–521.

[134] E. Argollo, A. Falcón, P. Faraboschi, M. Monchiero, and D. Ortega, "COTSon: Infrastructure for Full System Simulation," *ACM SIGOPS Operating Systems Review*, vol. 43, no. 1, pp. 52–61, Jan. 2009.

[135] N. Hardavellas, S. Somogyi, T. F. Wenisch, R. E. Wunderlich, S. Chen, J. Kim, B. Falsafi, J. C. Hoe, and A. G. Nowatzyk, "SimFlex: A Fast, Accurate, Flexible Full-system Simulation Framework for Performance Evaluation of Server Architecture," *ACM SIGMETRICS Performance Evaluation Review*, vol. 31, no. 4, pp. 31–34, Mar. 2004.

[136] F. J. Ridruejo, J. Miguel-Alonso, and J. Navaridas, "Full-system Simulation of Distributed Memory Multicomputers," *Cluster Computing*, vol. 12, no. 3, pp. 309–322, Sep. 2009.

[137] "Standard Performance Evaluation Corporation. SPECweb99 design document." [Online]. Available: http://www.spec.org/web99/docs/whitepaper.html

[138] "Hewlett-Packard Company. Netperf: A network performance benchmark." [Online]. Available: http://www.netperf.org

[139] N. Agarwal, T. Krishna, L. S. Peh, and N. K. Jha, "GARNET: A Detailed On-Chip Network Model inside a Full-System Simulator," in *2009 IEEE International Symposium on Performance Analysis of Systems and Software*, April 2009, pp. 33–42.

[140] S. Charles, C. A. Patil, U. Y. Ogras, and P. Mishra, "Exploration of memory and cluster modes in directory-based many-core cmps," in *2018 Twelfth IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, Oct 2018, pp. 1–8.

[141] A. Gutierrez, J. Pusdesris, R. G. Dreslinski, T. Mudge, C. Sudanthi, C. D. Emmons, M. Hayenga, and N. Paver, "Sources of error in full-system simulation," in *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2014, pp. 13–22.

[142] A. Butko, R. Garibotti, L. Ost, and G. Sassatelli, "Accuracy evaluation of gem5 simulator system," in *7th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*. IEEE, 2012, pp. 1–7.

[143] A. Akram and L. Sawalha, "A comparison of x86 computer architecture simulators," 2016.

[144] A. Akram and L. Sawalha, "86 computer architecture simulators: A comparative study," in *2016 IEEE 34th International Conference on Computer Design (ICCD)*, Oct 2016, pp. 638–645.

[145] "International Technology Roadmap for Semiconductors. (2010). Process Integration, Devices, and Structures Update," http://www.itrs.net/, [Online].

[146] "International Technology Roadmap for Semiconductors. The Model for Assessment of CMOS Technologies and Roadmaps (MASTAR)," http://www.itrs.net/models.html, [Online].

[147] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *Proceedings of the INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE*, 1995, pp. 24–36.

[148] M. Müller, D. Charypar, and M. Gross, "Particle-based Fluid Simulation for Interactive Applications," in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, 2003, pp. 154–159.

[149] G. Grahne and J. Zhu, "Fast Algorithms for Frequent Itemset Mining using FP-Trees," *IEEE transactions on knowledge and data engineering*, vol. 17, no. 10, pp. 1347–1362, 2005.

[150] D. Heath, R. Jarrow, and A. Morton, "Bond Pricing and the Term Structure of Interest Rates: A Discrete Time Approximation," *Journal of Financial and Quantitative Analysis*, vol. 25, no. 4, pp. 419–440, 1990.

Publications Related to Thesis

Book Chapter:

- **S. Agarwal**, and H. K. Kapoor, "Lifetime Enhancement of Non-volatile Caches by Exploiting Dynamic Associativity Management Techniques," *VLSI-SoC: The Internet of Things: SoC Opportunities and Challenges, IFIP Advances in Information and Communication Technology, Springer*, Volume 500, pp. 46–71, May 2019.

Journal(s):

- **S. Agarwal**, and H. K. Kapoor, "Reuse-Distance-Aware Write-Intensity Prediction of Dataless Entries for Energy-Efficient Hybrid Caches," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Volume 26, Oct 2018, Pages 1881-1894.
- **S. Agarwal**, and H. K. Kapoor, "Improving the Lifetime of Non-Volatile Cache by Write Restriction," *IEEE Transactions on Computers*, Volume 68, Sept 2019, Pages 1297-1312.

Under Review:

- **S. Agarwal**, and H. K. Kapoor, "Improving the Performance of Hybrid Caches using Partitioned Victim Caching," *ACM Transactions on Embedded Computing Systems*. (Major Revision Submitted)

Conferences:

- **S. Agarwal** and H. K. Kapoor, "Enhancing the Lifetime of Non-volatile Caches by Exploiting Module-Wise Write Restriction"-*29th ACM Great Lakes Symposium on VLSI (GLSVLSI '19)*, 2019, pp. 213-218, Washington, D.C., USA.

- **S. Agarwal**, and H. K. Kapoor, "Targeting Inter Set Write Variation to Improve the Lifetime of Non-volatile Cache using Fellow Sets"-*25th IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC '17)*, 2017, pp. 1-6, Abu Dhabi, UAE. **(Best Paper Award)**

- **S. Agarwal** and H. K. Kapoor, "Towards a Better Lifetime for Non-volatile Caches in Chip Multiprocessors"-*30th International Conference on VLSI Design (VLSID '17)*, 2017, pp. 29-34, Hyderabad, India.

- **S. Agarwal** and H. K. Kapoor, "Restricting Writes for Energy-Efficient Hybrid Cache in Multi-Core Architectures"-*24th IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC '16)*, 2016, pp. 1-6, Tallinn, Estonia.

Under Review:

- "Write Latency Aware Victim Caching for Caches made using Non-volatile Memory".

Other Publications of the Author

Journal:

- A.Nath, **S. Agarwal**, and H. K. Kapoor, "Reuse Distance based Victim Cache for Effective Utilisation of Hybrid Main Memory System" *ACM Transactions on Design Automation of Electronic Systems*. (Accepted)

Conference:

- S. S. Manohar, **S. Agarwal** and H. K. Kapoor, "Towards Optimizing Refresh Energy in embedded-DRAM Caches using Private Blocks"-*29th ACM Great Lakes Symposium on VLSI (GLSVLSI '19)*, 2019, pp. 225-230, Washington, D.C., USA.

- K. Rani, **S. Agarwal** and H. K. Kapoor "Non-blocking Gated Buffers for Energy Efficient on-chip Interconnects in the era of Dark Silicon"-*8th International Symposium on Embedded computing and system Design (ISED '18)*, 2018, pp. 74-79, Kochi, India. **(Best Paper Award)**

- A. Kulkarni, C. Joshi, K. Rani, **S. Agarwal**, S. P. Mahajan and H. K. Kapoor, "Towards Analysing the Effect of Snoozy Caches on the Temperature of Tiled Chip Multi-Processors"-*8th International Symposium on Embedded computing and system Design (ISED '18)*, 2018, pp. 230-235, Kochi, India.

- A. Kulkarni, K. Rani, **S. Agarwal**, S. P. Mahajan and H. K. Kapoor, "Towards Analysing the Effect of Hybrid Caches on the Temperature of Tiled Chip Multi-Processors"-*4th International Symposium on Smart Electronic Systems (iSES '18)*, 2018, pp. 52-57, Hyderabad, India.

- S.Priya, **S. Agarwal**, and H. K. Kapoor, "Fault Tolerance in Network on Chip Using Bypass Path Establishing Packets"-*31st International Conference on VLSI Design (VLSID '18)*, 2018, pp. 457-458, Pune, India.

- **S. Agarwal** and H. K. Kapoor, " Towards a dynamic associativity enabled write prediction based hybrid cache". *20th International Symposium on VLSI Design and Test (VDAT '16)*, 2016, pp. 1-6, Guwahati, India.