Performance and Lifetime Enhancement of Non-Volatile Memory Caches

Sivakumar S.

Performance and Lifetime Enhancement of Non-Volatile Memory Caches

Thesis submitted in partial fulfilment
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science and Engineering

by

Sivakumar S.

Under the supervision of

Dr. John Jose

Department of Computer Science and Engineering
Indian Institute of Technology Guwahati

January 2025

To my family, teachers, and friends

- who have always believed in and supported me.

Declaration

I hereby certify that

a. The work contained in this thesis is original and has been done by myself and the general supervision of my supervisor.

b. The work has not been submitted to any other institute for any degree or diploma.

c. Whenever I have used materials (data, theoretical analysis, results) from other sources, I have given due credit by citing them in the text of the thesis and giving their details in the references. Elaborate sentences used verbatim from published work have been clearly identified and quoted.

d. No part of this thesis can be considered plagiarism to the best of my knowledge and understanding and take complete responsibility if any complaint arises.

Date : **19** / **01** / **2025** **Sivakumar S.**

Place: Guwahati, India

भारतीय प्रौद्योगिकी संस्थान गुवाहाटी
INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI
Department of Computer Science & Engineering

Dr. John Jose
Associate Professor
johnjose@iitg.ac.in
http://www.iitg.ac.in/johnjose/

Thesis Certificate

This is to certify that the thesis entitled **Performance and Lifetime Enhancement of Non-Volatile Memory Caches** being submitted by **Sivakumar S.** to the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati, is a record of bonafide research work carried out by him under my supervision and is worthy of consideration for the award of the degree of Doctor of Philosophy of the Institute.

To the best of my knowledge, no part of the work reported in this thesis has been presented for the award of any degree at any other institution.

Date : **19** / **01** / **2025**

Place: Guwahati, India

Dr. John Jose
(Thesis Supervisor)

Acknowledgments

I embarked on my journey at IIT Guwahati in 2016 as a Project Engineer under Dr.John Jose, whose guidance and encouragement inspired me to pursue a PhD. His unwavering belief in my potential and his advice motivated me to explore the world of research. In June 2017, I was selected for the PhD program, and this decision marked a turning point in my life.

Dr. John Jose has been more than just a supervisor; he has been a true mentor. He provided me with the freedom to explore my research interests and offered numerous opportunities to develop both my technical and non-technical skills. My first international travel experience was under his guidance, an experience that I will cherish forever. I am deeply grateful for his continuous support, mentorship, and invaluable experiences that have shaped my academic journey.

In December 2017, I attended a GIAN course at IIT Madras by Prof. Vijaykrishnan Narayanan from the Department of Computer Science and Engineering at Pennsylvania State University. His insightful talk sparked my interest in exploring Non-Volatile Memories. I had the opportunity to interact with him again during his visit to IIT Guwahati in August 2022. Despite his busy schedule, Prof. Narayanan generously provided regular guidance and critical inputs, which eventually led to a publication with him. I sincerely thank him for his support and look forward to future collaborations.

I want to thank my doctoral committee members: Chairperson Prof. Jatindra Kumar Deka, Dr. Aryabartta Sahu, and Dr. Moumita Patra. Their valuable inputs and suggestions throughout the course of my PhD were instrumental in shaping the direction of my research. I want to extend my gratitude to all teaching and non-teaching staff of the Department of Computer Science and Engineering, especially Prof. T. Venkatesh and Prof. Sukumar Nandi for their help and support.

The inception of the MARS (Multicore Architecture Research and Systems) Lab in 2016 was another milestone in this journey. I was honoured to be among its first members alongside Abhijit, Dipika, and Manjari. The camaraderie and collaboration within the lab were instrumental in my development. Abhijit and Dipika's inputs and guidance were not only relevant but continue to motivate and inspire me to this day. Manju and Amit, who joined us later, have supported us. Manju, like a caring elder sister, was always concerned about my well-being, while Amit, though silent, has been accommodating and trustworthy. In the final phase of my PhD, the support of the new "Martians" – Syam, Vivekananda, and Rajeswari – was invaluable. Their

assistance, encouragement, and camaraderie helped me navigate the challenges of this period, making the journey more manageable. My campus life at IIT Guwahati was memorable, thanks to our group "Kameng Gadagam", which we formed based on our hostel. Al Ameen and Abdul Khader have always been with me; even now, they remain like brothers. I also fondly remember other members of the group – Dr Vivek Venugopal, Muhammad Shafeeque, Kiran, Joe Augustine, Ann Sheryl, and Albin – who made my stay in Guwahati a funny and unforgettable experience.

I am deeply grateful to all my teachers from school to college, who have played a crucial role in shaping my academic journey. I want to mention Prof. Rekha James from Cochin University of Science and Technology, who motivated and supported me in pursuing a PhD, and Mrs. Jyothi Sukumaran, who first taught me electronics at the school level and sparked my interest in the subject. My best friends, Arun Antony, Francis Amal, and Vishnu Lal, have been my constants for over two decades, always there for me in my ups and downs. Our enduring friendship has been a source of strength and joy. Finally and most importantly, I express my deepest gratitude to my parents, who always believed in me and provided the best possible support. I would like to thank my understanding brother and my wife, Divya, who stood firmly by my side through all the ups and downs during these years. Thank you all for making my life a good one.

To all those who have been part of this journey, whether mentioned here or not, your contributions have been significant, and I extend my heartfelt gratitude. This thesis is a testament to the collective efforts, support, and encouragement I have received throughout this remarkable journey.

Guwahati, January 2025 **Sivakumar S.**

Abstract

In recent decades, technological advancements have led to the emergence of numerous applications requiring increased computing power and larger on-chip and off-chip memory capacities. However, the memory technologies are not scaling up with the computational throughput of modern multi-core processors. Due to their low packaging density and high leakage power, traditional memory technologies like SRAM and DRAM face challenges in meeting substantial on-chip memory demands. Researchers have developed alternative solutions to address the growing need for memory, such as emerging non-volatile memory technologies like STT-RAM, PCM, and ReRAM. These techniques have the advantages of high packaging density, low power consumption and non-volatility. NVMs can be realized as Single Level Cells (SLC) or Multi Level Cells (MLC). SLCs store one bit of information per cell, whereas MLCs can store more than one bit per memory cell. The package density of NVMs can be further improved by using MLC NVMs instead of SLC NVMs.

Despite their advantages, these memory technologies have limited write endurance, high write latency and high write energy consumption. This highlights the necessity for policies that reduce write operations or evenly distribute them across memory cells, extending the lifetime of memory by mitigating premature wear-out caused by frequent writes. Considering the limitations of NVMs that need to be addressed in order to use them as cache memory, the thesis proposes optimization techniques for SLC and MLC NVM caches. The proposed techniques perform better than existing lifetime improvement techniques for SLC and MLC NVMs. The thesis shows that the proposed technique extends lifetime of MLC NVMs comparable to that of SLC NVMs with less area overhead.

Table of Contents

List of Figures

Page

List of Tables

Page

List of Acronyms

Acronym	Expansion
IoT	Internet of Things
NVM	Non Volatile Memory
STT-RAM	Spin Transfer Torque Random Access Memory
PCM	Phase Change Memory
ReRAM	Resistive Random Access Memory
SRAM	Static Random Access Memory
DRAM	Dynamic Random Access Memory
IPC	Instructions Per Cycle
LLC	Last Level Cache
SLC	Single Level Cell
MLC	Multi Level Cell
DM	Direct Mapping
CSM	Cell Split Mapping
AMAT	Average Memory Access Time
MPKI	Misses Per Kilo Instructions
WPKI	Writes Per Kilo Instructions
SWWR	Static Window Write Restriction
DWWR	Dynamic Window Write Restriction
ETB	Embedded Trace Buffer
SoC	System on Chip

Chapter

1

Introduction

The amount of data we process daily has increased significantly over the last few decades. The rapid rise of IoT, on-demand video platforms, and data-intensive applications has drastically increased demand for high computing power and memory for end devices. The processor works at a higher clock speed compared to memory. To mitigate the speed mismatch between processor and memory, computing systems employ cache memory, a high-speed small memory near the processing element. Popular applications such as high-performance computing, gaming applications, and video streaming platforms exhibit spatial or temporal locality, or both. The spatial locality of an application refers to the tendency of an application to access contiguous memory locations in a short period. Matrix multiplication, video/audio streaming, and web browsing are popular applications that show spatial locality. On the other hand, the tendency of an application to access the same memory location for a short period is termed temporal locality. Using cache memory to keep frequently accessed blocks improves system performance by reducing memory access time, thereby improving CPU utilization. The small size of cache memories limits the blocks stored in them. The applications with large memory footprint demand larger on-chip/off-chip caches for better system performance. Unfortunately, conventional memory technologies

such as SRAM and DRAM are inadequate to meet the demand for large volumes of on- and off-chip memories, as they occupy a lot of space and dissipate more static power [1]. This led to a search for alternative technologies to replace them in various levels of memory hierarchy and non-volatile memory technologies. Spin Transfer Torque Random Access Memory (STT-RAM) [2] [3], Phase Change Memory (PCM) [4] and Resistive RAM [5] were found to be promising candidates. Although these technologies are more compact and have non-volatility, they have limitations, such as low write endurance, high write energy and latency. These issues are more severe in the case of Multi-Level Cell (MLC) NVMs than Single Level Cell (SLC) NVMs. Applications with non-uniform write patterns can cause some portions of memory to be more heavily written than others, which results in the early wearing out of corresponding non-volatile memory cells, owing to its limited write endurance. In the following sections of the thesis we discuss, how to make use of NVM techniques to realize cache memories using various optimization techniques in detail.

1.1 Thesis Motivation

Over the past few decades, the demand for on-chip memory has increased due to data-intensive and compute-intensive applications, and the popular, conventional memory technologies are inadequate to meet this demand.

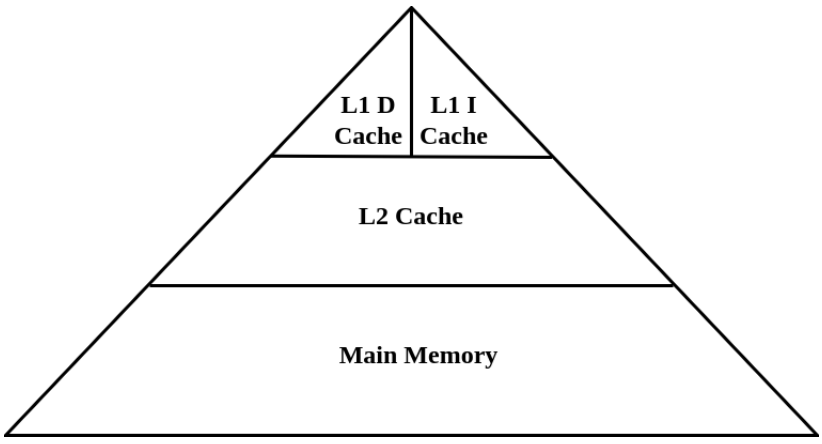

Figure 1.1: Memory Hierarchy

Figure 1.1 shows the memory hierarchy of a system with two levels of cache and

main memory. The size and latency increase as we move from L1 to the main memory in the memory hierarchy. L1 and L2 caches are realized using SRAM cells. L1 cache generally has a split architecture with separate space reserved for storing instructions and data blocks. L2 cache generally follows a unified architecture in which memory cells are not distinguished between instruction and data blocks. Realizing large LLCs (L2 or L3) using SRAM is challenging as SRAM cells occupy more on-chip space and dissipate more leakage power. So, we need better technology to realize large LLCs with less area and power overhead. As mentioned before, NVMs have the advantage of higher packaging density, low leakage power, and non-volatility compared to conventional SRAM and DRAM technologies. However, their low write endurance, high write energy and latencies are serious challenges that must be addressed when implemented as caches. As previously mentioned, applications without a well-distributed memory footprint may lead to frequent write operations concentrated on a few memory cells, resulting in premature wear out of memory cells. The early deterioration of specific memory cells can adversely impact the overall performance, reliability, and lifespan of the memory unit. A straightforward solution to address this issue is implementing a technique to ensure a uniform distribution of writes across the memory, irrespective of the application's memory footprint and is referred to as wear-leveling [1] [6]. Write minimization is another approach which minimize the number of writes to NVMs, thereby extending their lifetime [1]. In cache memories, variations in write operation can occur at different levels. Intra-set write variation refers to the variation within a set of cache memory, while inter-set write variation pertains to the variation across different sets. These variations are quantified using the coefficients of intra-set ($IntraV$) and inter-set ($InterV$) variation, as given in the equations below [6].

$$IntraV = \frac{1}{N \cdot Write_{avg}} \sum_{k=1}^{N} \sqrt{\frac{\sum_{l=1}^{M} \left(W_{k,l} - \sum_{m=1}^{M} \frac{W_{k,m}}{M}\right)^2}{M-1}} \tag{1.1}$$

$$InterV = \frac{1}{Write_{avg}} \sqrt{\frac{\sum_{k=1}^{N} \left(\sum_{l=1}^{M} \frac{W_{k,l}}{M} - W_{avg}\right)^2}{N-1}} \tag{1.2}$$

Where N is number of sets in cache.

M is the number of ways in a set.

$W_{k,l}$ is the write count in set k and way l.

$Write_{avg}$is average write count given by

$$Write_{avg} = \frac{\sum_{k=1}^{N} \sum_{l=1}^{M} W_{k,l}}{N \cdot M} \tag{1.3}$$

Write variation is a critical issue in designing cache or memory subsystems with

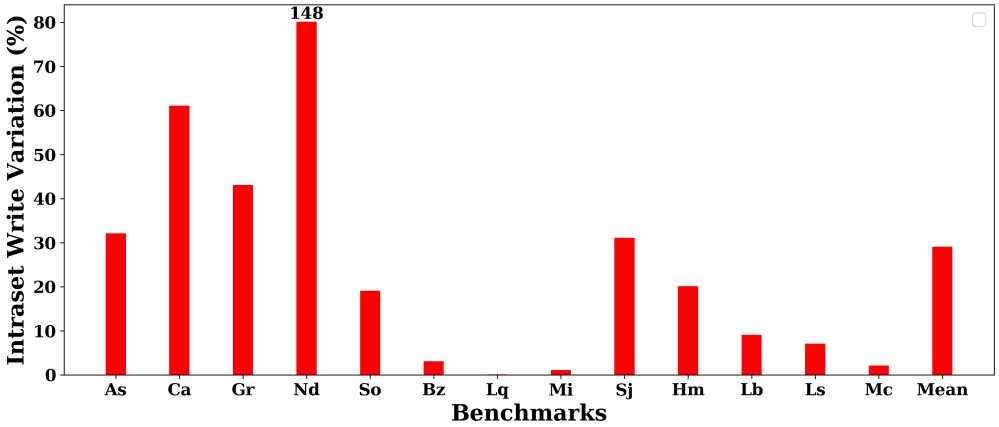

Figure 1.2: Intra-set write variation for L2 cache for various benchmarks

limited write endurance. Significant write variation can severely impact the product's lifetime, as a small subset of memory cells experiencing the highest write traffic can lead to the failure of the entire cache or memory subsystem, even if most cells remain far from wear-out. A large value of $InterV$indicates that cache lines in different sets experience vastly different write frequencies, often caused by applications with skewed address residency. Similarly, large $IntraV$ value arises when one cache line within a set frequently receives write hits, absorbing a disproportionate number of writes, leaving the remaining $M-1$ lines in an M way associative cache with uneven write distribution.

Inter-set write variation can be mitigated by dynamic reallocation, where data is moved between sets, or by improving address mapping schemes [7] to balance write traffic across sets. However, addressing intra-set write variation is more challenging because the cache replacement policies (e.g., LRU) are designed to manage temporal

locality, not to distribute writes evenly across lines within a set. When a single cache line in a set wears out, the entire set may become unusable due to associative conflicts or invalid entries. This reduces the overall capacity of the cache and accelerates the degradation of the memory subsystem, especially in technologies with limited write endurance, like NVMs. Hence, intra-set write variation is a much more severe issue. The enhancement techniques discussed in this thesis primarily focus on improving the lifetime and performance of NVM caches by reducing the intra-set write variation.

The variation in writes within the sets of an L2 cache memory is illustrated in Figure 1.2, depicting various benchmark programs from the SPEC CPU2006 benchmark suite [details of the benchmarks are given in Chapter 3]. A higher intra-set variation value suggests that certain blocks within a cache memory set experience more write accesses than others. This result emphasizes the importance of employing techniques to prevent repeated writes to specific blocks, either by bypassing them or distributing them throughout the memory.

1.2 Thesis Contributions

As discussed earlier, incorporating NVMs in on-chip caches offers significant advantages. First, compared to SRAM and DRAM, NVMs provide denser storage due to their smaller cell size. This enables substantially larger caches, leading to lower cache miss rates and improved performance compared to SRAM-based caches. Second, NVM caches can significantly reduce energy consumption. Studies have shown that caches account for up to 50% of a microprocessor's energy usage [8], with leakage energy comprising as much as 80% of the total cache energy consumption [9]. By eliminating leakage energy in standby mode, NVMs can help to lower overall energy usage. However, write variation poses a major challenge in designing cache or memory subsystems with limited write endurance. Severe write variation can drastically reduce the product's lifetime, as a small fraction of memory cells exposed to high write traffic can render the entire cache or memory subsystem inoperative, even if most cells remain far from wear-out. By exploiting this vulnerability, attackers can create malicious applications that reportedly write on one or few memory locations,

resulting in early wear out. Consequently, using NVMs as last-level caches (LLCs) without optimization techniques is challenging.

This section briefly summarizes our three proposed techniques that address this issue, enhancing the lifetime and performance of NVM caches. Figure 1.3 highlights the three main contributions of this thesis: two-lifetime improvement techniques tailored for SLC NVM LLCs and one combined lifetime and performance enhancement method for MLC NVM LLCs. Detailed explanations and analyses of these techniques are presented in the subsequent chapters.

In this thesis, we analyze the lifetime improvement and write distribution (intra-set variation) achieved by our proposed techniques, which aim to ensure uniform write distribution and prevent repeated writes to memory. A reduction in intra-set variation inherently improves write distribution and reduces the system's susceptibility to malicious repeated write attacks. Therefore, we do not explicitly discuss detailed mitigation strategies for such attacks.

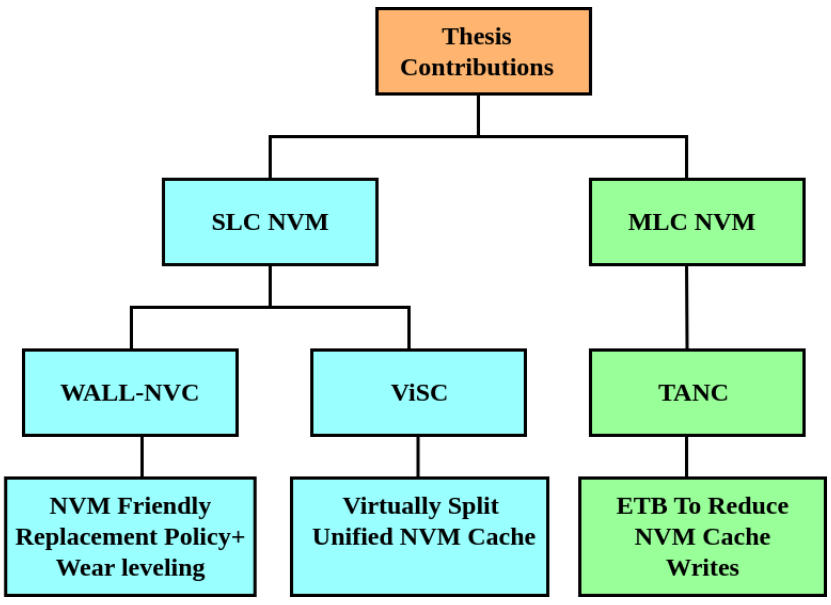

Figure 1.3: Overview of Thesis Contributions

1.2.1 Write Aware Last Level Non-Volatile Caches (WALL-NVC)

WALL-NVC is a promising lifetime enhancement technique for SLC NVM LLCs. Unlike the state-of-the-art lifetime improvement techniques for SLC NVM caches [10] [11] [12] [13] [14], WALL-NVC focuses on the impact of the cache replacement policies on the cache lifetime and implements a customized replacement policy for SLC NVM caches. Conventional cache block replacement strategies, such as Least Recently Used (LRU), cause the cache to wear down faster, as some cache blocks are retrieved more frequently than others. The thesis discusses Write Aware Last Level Non-Volatile Cache (WALL-NVC) for extending the lifetime of NVM while executing applications with non-uniform writes and to safeguard against targeted malicious attacks by repeatedly writing to certain blocks as its first contribution.

WALL-NVC is a dual-stage wear-leveling approach, contrasting with most modern wear-leveling methods. Choosing a better victim block for cache replacement in NVMs is handled by a new LRU-CB replacement policy, which is the first stage. The second stage uses a conventional write distribution technique with LRU-CB to extend the NVM lifetime. WALL-NVC is a reactive approach that uses counters at various levels of memory hierarchy to activate the write distribution mechanism. The Chapter 4 discusses the detailed description and analysis of WALL-NVC.

1.2.2 Virtually Split Last Level Non-Volatile Cache

Most of the modern processors use private split cache architecture for L1 caches and shared unified design for the subsequent levels of caches. The data blocks are more frequently accessed for writes than the instruction blocks. When NVMs are utilised in L1 caches, D-cache ages out more quickly than I-cache. Experimental studies indicate that in an L1 split cache, the D-cache experiences an average of 472x more writes than the I-cache [15]. This significant write variation between I and D caches indicates a significant variance in write access between instruction and data blocks. As stated before, data and instructions are stored together in unified last-level caches. Compared to the number of write operations to the blocks that store instructions,

we observe that the write count of the blocks that store data is high. By virtually splitting unified LLC for wear-leveling, our proposed approach, ViSC (Virtually Split Cache), explores the write variance across the data and instruction blocks.

ViSC logically divides the set of n-way set associative cache into m instructions ways and m-n data ways. The m instruction ways are exclusive for instructions and remaining m-n are exclusive for data. To distribute the writes uniformly, the logical mapping of LLC blocks into instructions and data is modified periodically. After every fixed time epoch, the blocks which are currently reserved for instruction will act as data blocks in next time epoch. Similarly, blocks which are reserved for data in current time epoch will act as instruction blocks. ViSC operates in three variants. The most proactive is the base version, which uses a static reorganisation interval and initiates reorganisation regardless of write pattern. The reorganisation interval is dynamically selected in the other two variants, E-ViSC and P-ViSC, based on the application write pattern. Chapter 5 of the thesis discusses the working and experimental analysis in detail.

1.2.3 Trace buffer Assisted Last Level Non-Volatile Cache

NVMs can be realized as SLC or MLC. MLC NVMs store two or more bits of data in each memory cell, in contrast to SLC NVMs, which can only store a single bit per memory cell. Despite having a much higher packing density than SLCs, MLC NVMs have short lifetime and large access time. MLC caches use 1.84x less space and 2.62x less leakage power than SLC caches for a given cache size. To improve the lifetime and performance of MLC LLCs the thesis proposes Trace buffer Assisted Non-volatile Memory Cache (TANC) as its third contribution, which utilizes a portion of the underutilized Embedded Trace Buffers (ETB) to reduce frequent writes to MLC NVM LLCs. ETBs are storage spaces available in modern processors for post-silicon validation but are left unused afterwards.

The LLC blocks which are frequently accessed for write operations are kept in ETB and all the read/write request to that particular block are serviced from ETB instead of LLC. When ETB is full, least recently written block replaced by writing back its contents to LLC. This reduces the number of writes to MLC NVM LLC,

which has less write endurance as most of the write operations are done on SRAM based ETB. Based on the selection of LLC blocks stored in ETB, TANC has different variants, and one variant addresses the thrashing effect of cache memories caused by poor temporal and spatial locality of applications. Chapter 6 discusses on TANC and its variants in detail.

1.3 Thesis Organization

The thesis is organized into seven chapters. The brief description of the subsequent chapters is given below.

- **Chapter 2** discusses about the background details of various NVM technologies, their characteristic features and challenges. This chapter also summarizes different state of the art techniques for life time and performance improvement of SLC and MLC NVM caches.
- **Chapter 3** discusses about the experimental setup used for analysis and comparison of the proposed techniques with other state of the art techniques. This chapter briefs about the simulator used for the experiment and benchmarks used for the analysis.
- **Chapter 4** discusses about the first contribution of the thesis, WALL-NVC a technique to enhance the lifetime of SLC NVM LLCs. The chapter discuss about working of the proposed technique, experimental setup, result analysis and comparison with other state-of-the-art lifetime improvement techniques for NVMs.
- **Chapter 5** gives a detailed analysis of second contribution of the thesis, ViSC and its variants. ViSC is lifetime enhancement technique for SLC NVM LLCs. ViSC is a low overhead lifetime enhancement technique compared to similar techniques. Results for various system configurations of the proposed techniques are also discussed in this chapter.
- **Chapter 6** is on the third contribution of the thesis, TANC which a novel lifetime enhancement technique for NVM LLCs. Unlike previous contributions

TANC is an enhancement technique customized for MLC NVMs. TANC is a low overhead technique as it utilizes unused resources for lifetime enhancement and the result analysis discussed in this chapter shows that the proposed technique improves the lifetime and performance of SLC NVMs.

- **Chapter 7** concludes the thesis and discusses few future works and extensions that can be done based on the contributions of the thesis.

CHAPTER

2

Background

In this chapter, we explore the functioning of emerging memory technologies, including Resistive RAM (ReRAM) [5], Phase Change Memory (PCM) [4], and Spin Transfer Torque RAM (STT-RAM) [1] [3], along with the challenges associated with their implementation as caches. Additionally, we review existing literature that provides a detailed discussion on state-of-the-art lifetime improvement techniques for NVM caches.

2.1 Emerging Memory Technologies

Emerging memory technologies, including STT-RAM, PCM and ReRAM are at the forefront of research and development in the quest for faster, more efficient, and more reliable memory storage solutions. These technologies offer unique advantages and characteristics, making them suitable for various applications.

Spin-transfer torque is a phenomenon where the orientation of a magnetic layer within a magnetic tunnel junction or spin valve can be altered by a spin-polarized current. This phenomenon finds application in flipping the active elements in magnetic random-access memory, known as Spin-Transfer Torque Magnetic Random-Access

Memory (STT-RAM or STT-MRAM). STT-RAM offers non-volatility and minimal leakage power consumption, a significant advantage over charge-based memories like SRAM and DRAM.

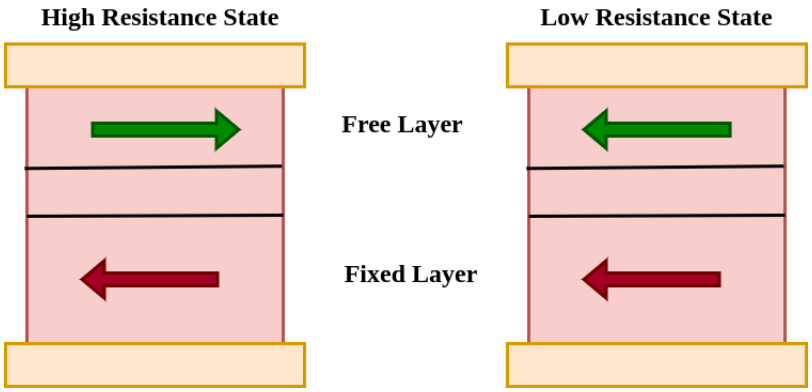

Figure 2.1: Schematic representation of an STT-RAM cell

As seen in Figure 2.1, STT-RAM, the second generation of Magneto Resistive RAM (MRAM) [2], uses the magnetic tunnel junction (MTJ) as a key component for bit information storage. An MTJ is made up of two ferromagnetic layers separated by a barrier layer. The fixed layer, also known as the reference layer, has a fixed magnetization direction, and the free layer, which can be changed by passing a spin-polarized current or an external magnetic field through the MTJ. When the reference layer and the free layer's magnetization directions are parallel, the MTJ displays a low-resistance state, signifying logical '0'. On the other hand, the MTJ assumes a high-resistance state when the two ferromagnetic layers' magnetization directions are anti-parallel representing logical '1'. STT-RAM also outperforms conventional MRAM in terms of lower power consumption and improved scalability, as MRAM relies on magnetic fields to switch the active elements.

A typical Phase Change Memory (PCM) cell is shown in Figure 2.2. The crystalline phase of the phase change material has low resistivity, while the amorphous phase has high resistivity. This significant resistivity difference is exploited by PCM. In PCM, the states "set" and "reset" stand for low and high resistance conditions, respectively. The processing temperature of the metal interconnect layers is high enough to crystallize the phase-change material, resulting in an initial low-resistance crystalline state. During a reset, a large electrical current pulse is applied for a

brief period, melting and rapidly quenching the programming region of the PCM cell to cause it to transition to the amorphous phase. This process results in an amorphous, highly resistive region within the PCM cell, affecting the overall resistance between the top electrode contact and the bottom electrode contact. To set the PCM cell back to the crystalline phase, a moderate electrical current pulse is applied to anneal the programming region at a temperature between the crystallization and melting temperatures, ensuring sufficient time for crystallization. Reading the state of the programming region involves measuring the cell's resistance using a small electrical current that does not disturb the current state. A Resistive Random Access

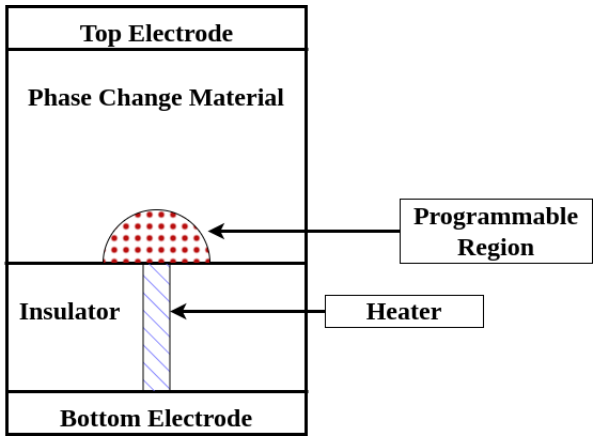

Figure 2.2: Schematic representation of a PCM cell

Memory (ReRAM) [5] comprises a memory cell with a resistive switching mechanism, featuring a metal-insulator-metal structure. This structure involves an insulating layer positioned between two metal electrodes. The schematic view of an ReRAM cell are depicted in Figure 2.3. Applying an external voltage pulse across the ReRAM cell facilitates a transition from a high resistance state or OFF state (logic value '0') to a low resistance state or ON state (logic value '1'), and vice versa. Initially, ReRAM is in the high resistance state. To shift the device to low resistance state, a high-voltage pulse, SET voltage, is applied and it results in formation of conductive paths in the switching layer, resulting in the ReRAM cell transitioning low resistance state (SET process). Conversely, to switch the ReRAM cell from low to high resistance state, a voltage pulse, referred to as the RESET voltage, is applied, facilitating this transition and is denoted as the RESET process. For efficient data reading from the ReRAM

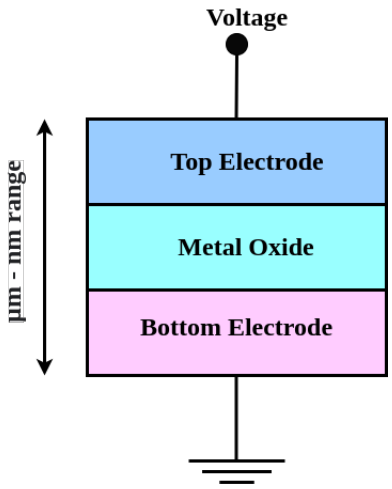

Figure 2.3: Schematic representation of a ReRAM cell

Table 2.1: Approximate values of device level properties for different memory technologies

	SRAM	DRAM	PCM	STT-RAM	Re-RAM
Cell size (F^2)	125-200	6-10	4-12	6-50	4-10
Access granularity	64 B	64 B	64 B	64 B	64 B
Read latency	1-10 ns	10-20ns	50-100 ns	10 ns	10-50 ns
Write latency	1-10 ns	10-20 ns	100-500 ns	10-100 ns	10-100 ns
Endurance (number of writes)	$> 10^{15}$	$> 10^{15}$	$10^8 - 10^9$	$> 10^{12}$	$> 10^{11}$
Standby power	60 nW	Refresh power	0	0	0

cell, a small read voltage, which does not disturb the current state of the cell, is applied. This voltage helps determine whether the cell is in a logic 0 or logic 1 state.

Approximate values of device level properties for different memory technologies are shown in Table 2.1. In comparison to SRAM and DRAM, emerging memory technologies exhibit significantly higher densities and comparable fast access times. Notably, non-volatile memory technologies possess zero standby power and are resistant to radiation-induced soft errors. Despite these advantages, the direct replacement of existing SRAM and DRAM technologies with the mentioned emerging memory technologies is challenging. For instance, SRAM and DRAM arrays are predominantly affected by leakage power. Conversely, while PCM or ReRAM arrays consume no leakage power when idle due to non-volatility, they demand considerably more energy during write operations. Consequently, exploring trade-offs in the utilization of diverse memory technologies at various hierarchy levels becomes a

crucial research area. Substituting SRAM-based on-chip cache with STT-RAM and PCM can enhance performance and decrease power consumption. The higher packing density of STT-RAM and PCM allows for a larger on-chip cache capacity, thereby reducing cache miss rates and improving overall performance. Zero standby leakage contributes to lowered power consumption. However, challenges associated with emerging memory technologies include the prolonged duration and increased energy consumption required for write operations. NVMs offer a more stable data storage mechanism than volatile SRAM and DRAM. Yet, directly replacing SRAM caches with PCM or STT-RAM caches may lead to performance drawbacks, particularly under high cache write intensity. Employing hybrid cache memory, buffers, and data compression becomes essential to mitigate the extended latency and elevated energy consumption in write operations for PCM or STT-RAM caches. Despite STT-RAM's higher density compared to SRAM, utilizing it directly as on-chip caches with frequent accesses proves impractical due to limited endurance. Wear-leveling is a technique widely used in NAND-flash memory, seeks to address write endurance limitations by evenly distributing write operations across storage cells, and this approach can also be applied to NVM caches and memory.

Among different NVM technologies, STT-RAM , with its superior write endurance and overall performance metrics, proves to be a more suitable choice for implementing last-level caches. In this thesis, we specifically adopt STT-RAM as our candidate to replace SRAM in cache memories, henceforth referring to NVM as STT-RAM. NVMs can be categorized as Single Level Cell (SLC) or Multi Level Cells (MLC). SLCs store one bit per memory cell, while MLCs can store two or more bits per memory cell. Figure 2.4 (a) illustrates the structure of SLC STT-RAM cell. An SLC STT-RAM cell consists of a Magnetic Tunneling Junction (MTJ) with free and reference ferromagnetic layers separated by an oxide layer, where the reference layer's magnetization direction is fixed and bits are stored by changing the free layer's magnetization direction. The MLC STT-RAM cell, can be either serial or parallel as shown in Figure 2.4 (b) and Figure 2.4 (c), respectively . In our work, we refer to MLC as serial MLC due to its superior performance and reliability compared to parallel MLCs. The serial MLC STT-RAM cell features two vertically stacked free

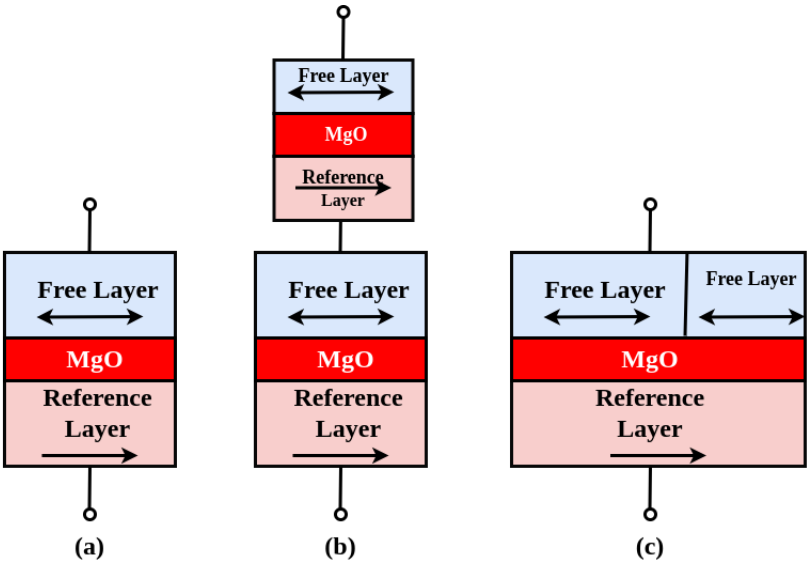

Figure 2.4: Schematic representation of (a) SLC (b) Serial MLC and (c) Parallel MLC STT-RAM cell

layers, while parallel MLC cells use a single MTJ with two independent free layers. In this context, the larger MTJ is termed a hard-bit with a high switching current, and the smaller MTJ, characterized by a lower switching current, is referred to as a soft-bit. Although MLC NVMs enhance data density, they exhibit higher lifetime, write energy, and latency compared to SLC. Soft-bit flipping is more straightforward than hard-bit flipping due to the smaller switching currents. Hard-bits, on the other hand, due to large switching currents, entail higher write energy and latency. The writes to hard-bits lead to the flipping of soft-bits, termed as write disturbance. Figure 2.5 illustrates how values are stored in the hard-bit (**X**) and the soft-bit (**Y**) of an MLC NVM cell, indicating possible state transitions and the retention of the current state. As illustrated in Figure 2.5, hard transitions cause the flipping of both hard- and soft-bits, while soft transitions only flip the soft-bit, leaving the hard bit unchanged. Therefore, to modify only the hard-bit, a two-step process is required: a hard transition followed by a soft transition. Cells can move from one state to another or retain its current state on following transitions, given below

- No transition: Current state of bits (hard and soft) are the same as new state (both bits retain current state)
- Soft transition: Change in current soft-bit state only whereas hard-bit retains

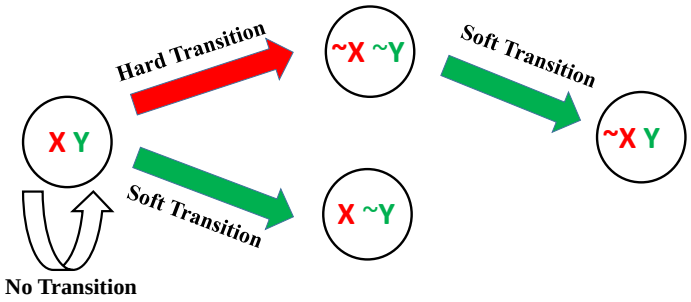

Figure 2.5: Hard and Soft-bit states during different transitions of a MLC STT-RAM cell

current state (small switching current)

- Hard transition: Change in hard-bit state. Soft-bit should retain/update its current state to updated hard-bit state (high switching current).

- Two-step transition: Change in a hard-bit only. Hard transition followed by soft transition.

The flipping of the soft-bit, denoted as Y to ∼Y, occurs in a single step as hard bits are unaffected by the small switching current. When a hard-bit transitions from X to ∼X, it leads to the simultaneous flipping of the associated soft-bit (from Y to ∼Y). However, due to this write disturbance, exclusive bit flips for hard-bits involve a two-step process. Initially, hard-bits undergo flipping using high switching currents, followed by the soft-bit being flipped back to its original state using a smaller switching current.

Since MLC NVMs have two different bits present in them, they can be configured to organize cache lines through in different approaches. Direct Mapping (DM) and Cell Split Mapping (CSM) as illustrated in Figure 2.6 are two popular stratagies for MLC NVM cache organizations. The organization of cache lines in the DM strategy involves a cache block comprising hard and soft bits, but it fails to leverage the faster accessibility of soft-bit cells. Conversely, in CSM, the hard bits of memory cells are grouped to form the hard-way, while all soft bits are grouped to form the soft-way.

CSM introduces variable latency for blocks based on their location. Read and write operations in the soft-way are streamlined into a single step. However, due to the potential for write disturbance, writing to the hard-way requires an initial read of the corresponding soft-way to safeguard its contents before completing the hard-way write. Experimental studies demonstrate that MLC based on CSM enhances system performance by 10.3% and reduces energy consumption by 26.0% [16] compared to conventional MLC STT-RAM and hence here after we consider CSM STT-RAM for our experiments and refer as MLC NVM.

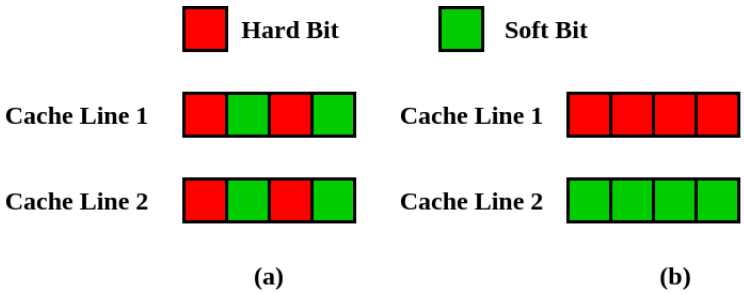

Figure 2.6: 4 bit cache lines in (a) Direct Mapping (b) Cell Split Mapping

NVMs are prone to premature wear-out caused by non-uniform write patterns and repeated write attacks [17] [18] due to the limited write endurance. Numerous state-of-the-art techniques have been developed to enhance the lifetime and performance of SLC and MLC NVMs. Next section delves into a detailed discussion of such advancements. While the improvement in the lifetime of SLC NVMs has been extensively explored, MLC NVMs face challenges due to their structural peculiarities. Implementing conventional SLC wear-leveling algorithms in MLC is considered challenging and not cost-effective.

2.2 Related Works

This section explores various techniques aimed at enhancing the lifetime and performance of SLC and MLC NVMs. Researchers across the globe have proposed diverse strategies to tackle the challenges associated with improving the endurance of NVM LLCs. These approaches include wear-leveling techniques [11] [13] [12] [14] [6], write reduction methods [19] [20], hybrid cache architectures [21] [10], and data

compression schemes [22]. Each of these methods addresses specific aspects of write variations and performance bottlenecks inherent in NVMs. In this section, we will delve into some of the most relevant and impactful works in detail, highlighting their contributions and limitations.

J. Wang et al. proposed i^2wap[6], a technique which leverages two global counters and registers. i^2wap incorporates swap-shift, a wear-leveling strategy minimizing cache inter-set write variations, and Probabilistic Set Line Flush (PoLF), a novel approach for mitigating intra-set write variations. Key idea behind PoLF is that the frequent access to hot data blocks, which remains unchanged in traditional replacement policies, causes write variations. PoLF probabilistically flushes hot data to optimize write distribution. In case of a cache write hit, instead of writing directly to the hit data block, the new data is placed in the write-back buffer, marking the cache line as invalid. Consequently, other cold data can replace the block containing the hot data, allowing the relocation of hot data to other areas. Sparsh Mittal et al. proposed EqualWrites [12] and EqualChance [11] techniques to reduce intra-set variation and improve the lifetime of non-volatile memory caches. EqualWrites identifies substantial intra-set write variation when the difference between the number of writes to two blocks in a cache set exceeds a threshold. By swapping data items in these blocks, intra-set write variation is reduced, leading to a better distribution of writes and enhanced cache life. The EqualChance mechanism periodically changes the block locations of data to distribute writes more uniformly across cache lines. To achieve this, it uses counters to track the number of writes for each set. After a certain threshold of writes is reached, hot (frequently written) data is swapped with cold (infrequently written) data. The swap candidates may either be invalid blocks, a process termed I-shifting, or clean blocks, referred to as C-shifting. Dirty blocks are excluded from the swapping process since they are likely to be frequently written and would not effectively reduce wear. Another state-of-the-art approach for extending the lifetime of NVM caches is the use of periodically interchangeable write-restricted window. Techniques like Static Window Write Restriction (SWWR) [14], Dynamic-Window Write Restriction (DWWR), and Dynamic Way Aware Write Restriction (DWAWR) [13] fall under this category. SWWR divides the cache into logical windows,

treating one as a write-restricted window during each interval, with writes redirected to other windows. The core concept of DWWR involves dividing the cache into m equally sized windows and using a different window during each predefined execution interval. Unlike SWWR, where the write-restricted window is chosen in a round-robin manner, DWWR selects the window based on a counter associated with each window. This counter tracks the number of writes during the previous interval (i.e., from the upper-level cache to the last-level cache). At the start of each interval, the window with the highest number of writes is designated as the write-restricted (or read-only) window. Once the interval ends, the next write-restricted window is determined based on the counter values, and this process continues throughout execution. To avoid selecting the same window consecutively, the counter for the current write-restricted window is reset at the end of each interval. DWAWR selects heavily written ways instead of fixed windows as write-restricted for a given interval, providing effective wear-leveling. As previously discussed, the MLC NVM exhibits a shorter lifetime compared to SLC due to write disturbance. To mitigate the impact of write disturbance on soft-ways, a mechanism is required to safeguard data in soft-ways during hard-way writes. One simplistic solution is the Immediate Restore Scheme (IRS), in which the corresponding soft-way is read and rewritten after the completion of the hard-way write operation to ensure data consistency. The Adaptive Restoration Scheme [23] for write disturbance and read disturbance comprises two schemes for addressing write and read disturbances in MLC NVMs. To tackle write disturbances, this technique overwrites the soft-bit lines, which are less likely to be read, accumulating potential writes to address read disturbance. The soft-bit line is restored during eviction from the higher cache level. Both approaches demonstrate performance enhancement and substantial energy savings for MLC STT-RAM caches. The self-adaptive wear-leveling technique, designed for MLC NVM, achieves wear leveling by balancing writes across memory cells using two mapping tables; a global directory and an on-chip SRAM cache that maintains recently accessed mappings.

Another strategy for minimizing read disturbances is the restore-free mode [24], involving two-step read/write operations for frequently used hard-ways in MLC STT-RAM cache. This technique allows the logical turning off of specific hard/soft

domains in an MLC by fixing their data to "0" or "1". In the restore-free mode, the corresponding soft-way of the frequently used hard-way is deactivated after invalidation, and its data is written to a lower-level memory. Although eliminating write disturbance restorations streamlines hard-way writes into a one-step process and reduces energy consumption, the energy required for hard-way writing remains higher than for soft-way writing.

The technique of Alternative Encoding [22] aims to reduce two-step transitions in MLC STT-RAM cache. It employs two 3-bit codes for each 2-bit data, providing flexibility to eliminate two-step transitions. However, this encoding method results in a larger data size, improving the lifetime of the MLC STT-RAM cache. Despite sacrificing some data density, the data density of the alternating encoding-based MLC STT-RAM cache still surpasses that of the SLC STT-RAM cache. P. Saraf et al. [25] proposed replacement policies like the Refresh Aware Replacement Policy (RFR) to improve the lifetime and performance of STT-RAM caches. The authors focus on reducing the maximum number of writes, global write variation, and the average number of writes to enhance the endurance of write-optimized STT-RAM caches. Write-optimized STT-RAM refers to a configuration of STT-RAM designed to reduce its inherently high write latency by lowering the thermal barrier of its MTJ cells. The thermal barrier is a measure influenced by the physical and material properties of the MTJ, such as planar area, magnetic parameters, and free layer thickness. Reducing the thermal barrier decreases the switching current and the write pulse width (the duration for applying the switching current), resulting in faster write operations. However, reducing the thermal barrier also leads to a shorter retention time, the duration for which data can be reliably stored in an STT-RAM cell without a random bit flip. Refresh mechanisms are employed to mitigate the shorter retention time in write-optimized STT-RAM. Blocks that remain in the cache beyond their retention period are rewritten to prevent data loss, effectively maintaining reliability. Refresh-aware cache replacement policies also prioritize evicting blocks about to expire over recently refreshed blocks to balance performance and endurance. The techniques for enhancing the lifetime of NVM caches discussed thus far can be categorized into reactive and proactive approaches. Reactive methods use cache monitoring tools,

such as counters, to trigger specific actions once predefined conditions (e.g., reaching a threshold value) are met. On the other hand, proactive techniques continuously maintain enhancement mechanisms active, regardless of the application's behavior or execution time window. Many of the previously mentioned approaches rely heavily on counters or additional circuitry, leading to increased overhead. Upon further analysis of these methods, i^2wap [6] is found to have a drawback: it may invalidate the Most Recently Used (MRU) blocks, causing more accesses to the main memory. EqualChance swaps write-intensive blocks within a cache set with invalid or clean blocks based on a write counter threshold, but this requires more write counters, contributing to greater area and power overhead. SWWR uses a round-robin window selection process, which does not consider the write counts of other windows within the cache set, potentially allowing heavily written windows (or ways) to accumulate in the cache bank and negatively affecting cache lifetime. DWWR lacks consideration for heavily written ways that may exist in lightly written windows, meaning that even if a window has a low write count, it could still contain some highly written ways that are neglected. Lastly, RFR, while effective for write-optimized caches, is less effective for unoptimized caches. These shortcomings prompted us to explore techniques that can improve both the lifetime and performance of NVM caches with minimal overhead. This thesis proposes the following approaches for SLC and MLC NVM LLCs:

- WALL-NVC, a reactive technique utilizing a custom cache replacement algorithm designed for SLC NVM caches to enhance their lifespan.
- ViSC, a proactive technique that involves logically splitting the SLC NVM-based unified LLC. It also introduces variants E-ViSC and P-ViSC, falling into the hybrid category, incorporating features from both proactive and reactive approaches.
- TANC, a technique focused on enhancing the lifetime and reducing the write latency of MLC-based NVM caches.

We discuss these techniques in detail in the subsequent chapters. Table 2.2 compares the key ideas of selected techniques discussed in this section along with the

proposed techniques. Detailed analysis is given in coming chapters.

Table 2.2: Overview of proposed techniques and state-of-the-art approaches

	Key feature	Supports SLC NVM	Supports MLC NVM
i^2wap	Probabilistic line flushing	Yes	No
EqualWrites	Write distribution using counters	Yes	No
EqualChance	Write distribution using counters	Yes	No
DWWR	Write restriction using set partitioning (windows)	Yes	No
Restore Free	Two step write reduction in MLC NVM caches	No	Yes
WALL-NVC	Write distribution using counter with NVM friendly replacement policy	Yes	No
ViSC	Write distribution using logical partitioning	Yes	No
TANC	Write reduction using Embedded trace buffer	Yes (Can be modified)	Yes

CHAPTER

3

Experimental Setup

This chapter discusses about the overview of experimental setup used for analyzing the proposed techniques. Detailed experimental details for each contributions are given in corresponding chapters.

3.1 Adopted simulation environment

Simulators are software tools that mimic the operation of real-world systems or processes in a controlled and reproducible environment. They are used for testing, analysis, training, and research without the risks or costs associated with real-world implementation. Simulators allow architects and engineers to test new designs and configurations before physical implementation, identifying potential issues and optimizing performance. They enable detailed performance analysis of various architectural components under different workloads, such as CPUs, memory hierarchies, and interconnects. Organizations can reduce the costs and risks associated with building and testing physical prototypes by simulating designs. They help to verify that architectural changes or enhancements meet the required specifications and performance targets.

We employ the gem5 simulator [26] to analyze and implement the proposed techniques. Widely embraced in both academic and industrial circles, the gem5 simulator originated from the merger of the m5 simulator [27] at the University of Michigan and the GEMS simulator [28] from the University of Wisconsin. The gem5 simulator is highly modular, allowing users to customize and extend the simulator to meet specific research needs. It supports multiple CPU models, memory systems, and interconnects. The gem5 provides detailed and accurate models of modern processor architectures, including in-order and out-of-order cores, detailed memory systems, and various interconnects. It supports multiple instruction set architectures (ISAs), including ALPHA, x86, ARM, MIPS, RISC-V, and SPARC, making it versatile for different research scenarios. The gem5 simulator can simulate entire systems, including operating systems and full applications, providing a comprehensive view of system behaviour. Users can easily configure and modify various architectural parameters, enabling experimentation with different design choices and performance trade-offs. It can be integrated with other simulation and analysis tools, enhancing its capability for comprehensive architectural studies.

In our experiments, we utilize Ruby to model memory. Ruby, the memory system simulator within gem5, is a modular framework designed to model and simulate detailed memory hierarchies in modern computer systems. Ruby's advantage lies in its ability to keep coherence protocol specifications distinct from replacement policies and cache index mapping, allowing separate specification of network topology from implementation. Additionally, Ruby is highly configurable in nature and facilitates rapid prototyping. As shown in Figure 3.1, Ruby integrates three key components:

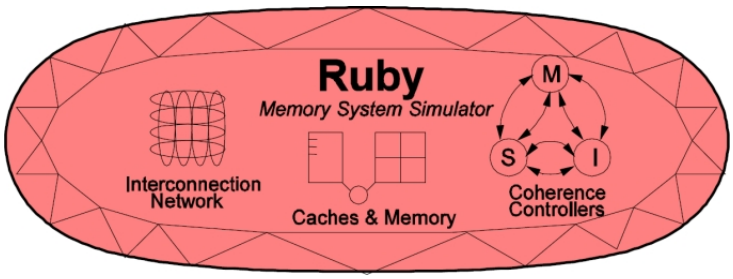

Figure 3.1: High-level view of the main components in Ruby [Image source: gem5 documentation (https://www.gem5.org)]

the interconnection network, which facilitates communication between processors, caches, and memory controllers using various topologies like mesh or bus; caches and memory, representing the hierarchical structure of caches (L1, L2, L3) and main memory, allowing detailed analysis of cache behaviours such as hits, misses, and replacement policies; and coherence controllers, which enforce consistency across caches using protocols like MSI (Modified, Shared, Invalid) or more advanced ones like MESI or MOESI. The coherence controllers ensure proper transitions between states (e.g., Modified, Shared, Invalid), while the interconnection network connects all components, enabling seamless data flow. The cache coherence in our setup is maintained using the MESI_Two_Level protocol [29], which features a two-level cache hierarchy with private L1 cache and shared L2 cache. In this arrangement, L1 and L2 maintain inclusion between them. At a higher level, the MESI_Two_Level protocol encompasses four stable states: M, E, S, and I. A block in the M state indicates that it is writable and has exclusive permission, having been dirtied as the only valid on-chip copy. The E state represents a cache block with exclusive permission (writable) but not yet written. S state signifies that the cache block is only readable, with multiple copies possibly existing in various private caches and the shared cache. The I state denotes that the cache block is invalid.

3.2 Benchmark Programs

For experimenting with our proposed architectures across different application categories, we utilize the SPEC CPU2006 benchmark suite [30]. The SPEC CPU2006 benchmarks, developed by the Standard Performance Evaluation Corporation (SPEC), are widely used in the evaluation of computer systems' CPU performance. SPEC CPU2006 includes diverse benchmarks designed to represent a wide range of applications. This ensures that the benchmarks are not biased toward any particular type of workload. The benchmarks are derived from actual applications and scientific computations, providing a realistic performance measure. The benchmarks are designed to be complex and compute-intensive, making them representative of real-world scenarios that require significant computational power. SPEC CPU2006 benchmarks

can be compiled and run on various computer architectures and operating systems, facilitating performance comparisons across different systems.

Table 3.1: Selected SPEC CPU2006 Benchmarks with arconyms and category

Name	Description	WPKI	MPKI
astar (As)	Path finding algorithms	Low	Mid
bzip2 (Bz)	Compression	Mid	High
calculix (Ca)	Structural Mechanics	Low	Low
gromacs (Gr)	Biochemistry/Molecular dynamics	Low	Low
h264ref (H2)	Video compression	Low	Low
hmmer (Hm)	Search gene sequence	High	Low
lbm (Lb)	Fluid dynamics	High	High
leslie3d (Ls)	Fluid dynamics	High	Mid
libquantum (Lq)	Physics : Quantum computing	Mid	High
mcf (Mc)	Combinatorial optimization	High	High
milc (Mi)	Physics: Quantum chromodynamics	Mid	High
namd (Nd)	Biology/Molecular dynamics	Low	Low
sjeng (Sj)	Artificial intelligence : chess	Mid	Mid
soplex (So)	Linear programming, optimization	Low	Mid

The benchmarks are provided as source code, allowing users to compile them with different compilers and optimization settings specific to their systems. SPEC CPU2006 specifies rigorous rules for conducting benchmark tests to ensure that results are comparable across different systems and configurations. The benchmarks are designed to produce consistent and repeatable results under the same conditions, making them reliable for performance evaluation. Table 3.1 provides details on the benchmarks used and acronyms associated with them.

We executed the benchmarks on a unicore system with a 512KB L2 cache using the gem5 simulator, recording the number of writes and misses over one billion instructions. From these measurements, we calculated Writes Per Kilo Instruction (WPKI) and Misses Per Kilo Instruction (MPKI). Benchmarks were categorized based on their WPKI to the last-level cache into Low (WPKI < 10), Mid ($10 \leq$ WPKI ≤ 30), and High (WPKI > 30), as shown in Table 3.1. Similarly, we classified benchmarks by MPKI into Low (MPKI $< 10\%$), Mid ($10\% \leq$ MPKI $\leq 60\%$), and High (MPKI $> 60\%$). This classification allows us to assess the effects of both existing

and proposed techniques on applications with different write characteristics.

The specific experimental setups for each technique are detailed in the respective sections of each chapter.

3.3 Performance Metrics

The key performance metrics used for experimental analysis of each thesis contributions are given below. Each of these parameters helps to compare the effectiveness of the techniques.

3.3.1 Intra and Inter Set Write Variation

Intra-set and inter-set write variations are two key performance metrics used to evaluate the lifetime enhancement techniques. Intra-set write variation refers to the variation within a set of cache memory, while inter-set write variation pertains to the variation across different sets. These variations are quantified using the coefficients of intra-set ($IntraV$) and inter-set ($InterV$) variation, as given in the equations below [6].

$$IntraV = \frac{1}{N.Write_{avg}} \sum_{k=1}^{N} \sqrt{\frac{\sum_{l=1}^{M} \left(W_{k,l} - \sum_{m=1}^{M} \frac{W_{k,m}}{M}\right)^2}{M-1}} \tag{3.1}$$

$$InterV = \frac{1}{Write_{avg}} \sqrt{\frac{\sum_{k=1}^{N} \left(\sum_{l=1}^{M} \frac{W_{k,l}}{M} - W_{avg}\right)^2}{N-1}} \tag{3.2}$$

Where N is number of sets in cache.
M is the number of ways in a set.
$W_{k,l}$ is the write count in set k and way l.
$Write_{avg}$is average write count given by

$$Write_{avg} = \frac{\sum_{k=1}^{N} \sum_{l=1}^{M} W_{k,l}}{N.M} \tag{3.3}$$

$IntraV$ and $InterV$ values reflect how well the writes are distributed across the cache memory. A low $IntraV$ and $InterV$ indicates that writes are equally distributed within and across the sets of the cache memory respectively. In an ideal scenario, where all

blocks have equal number of write count will result in $IntraV$ and $InterV$ values to be zero. Low value of write variation indicates that writes are not concentrated in few memory blocks and hence reducing the chances of early wearing out of memory cells.

3.3.2 Relative Lifetime

Relative lifetime (RL) of a given architecture with respect to the baseline architecture is given by

$$RL = \frac{Maximum\,number\,of\,writes\,in\,baseline\,architecture}{Maximum\,number\,of\,writes\,in\,given\,architecture} \tag{3.4}$$

Relative lifetime is popular metric used to analyse the effectiveness of lifetime enhancement techniques. We have utilized raw cache lifetime as it offers valuable insights and serves as the foundation for error-tolerant lifetime[11]. The results are presented for both the maximum number of writes on any block and intra-set variation (IntraV). The former focuses on the worst-case scenario for writes on a single block, while the latter accounts for the average number of writes and considers all blocks within the cache. Together, these metrics provide a comprehensive evaluation of a technique. Notably, these metrics have also been employed in other research studies [6], [31] [14] [13] [32]. Baseline refers to the un-optimized cache memory. Large value of relative lifetime indicate that given architecture is able to reduce the maximum write count.

3.3.3 Cache Hit Rate

Cache hit rate is a performance metric used in computing to measure the efficiency of a cache system. It represents the percentage of cache accesses that result in a hit, which means that the requested data were found in the cache. The higher the cache hit rate, the more effectively the cache serves the requested data without retrieving it from a slower level in the memory hierarchy [33].

$$Cache\,Hit\,Rate = \frac{Number\,of\,Cache\,Hits}{Total\,Number\,of\,Cache\,Accesses} \times 100 \tag{3.5}$$

3.3.4 Average Memory Access Time

Average Memory Access Time (AMAT) is a performance metric which measures the average time taken to access a memory location, considering both cache and main memory accesses [34]. AMAT provides insight into the overall speed at which data can be retrieved from the memory system and is crucial for understanding and optimizing the performance of system architectures. AMAT is calculated as

$$AMAT = Hit\,Time + Miss\,Rate \times Miss\,Penalty \tag{3.6}$$

Where

- Hit Time : The time it takes to access the data in the cache.
- Miss Rate : The fraction of memory accesses that result in a cache miss
- Miss Penalty: The additional time required to fetch the data from the next level of the memory hierarchy when a cache miss occurs.

CHAPTER

4

Write Aware Last Level Non-Volatile Cache

This chapter proposes a novel wear-leveling technique for improving the lifetime of SLC NVM caches. Applications with non-uniform write patterns are a serious concern for NVM caches as they might access some memory locations for write operation more frequently than others, leading to the early wear-out of memory cells. In this chapter we propose a two-stage technique that shows significant improvement in the lifetime of NVM LLC compared to state-of-the-art techniques.

4.1 Introduction

As discussed in the previous sections, applications with non-uniform write patterns can cause some cache memory cells to wear out faster than others. The lack of write-aware cache replacement policies can lead to frequent writes to specific cache blocks, causing those memory cells to wear out earlier than expected. These issues underscore the need for a system that minimizes or evenly distributes writes when using non-volatile memories at various levels of the memory hierarchy. We propose a

Write Aware Last Level Non-Volatile Cache (WALL-NVC) that extends the lifespan of non-volatile memory when used as a last-level cache. EqualWrites [12], a state-of-the-art wear-leveling technique, reduces intra-set write variation and improves lifespan by comparing LRU and random replacement policies for cache blocks. Conventional cache replacements consider only the recency of use while selecting the victim block. However, these blocks may have been heavily written in the past, and replacing them with a new block might speed up the wear of those memory cells as the probability of write access to a recently replaced block is higher than that of an existing block. Hence, these policies need to be optimized for NVMs. Replacement strategies like the Refresh Aware Replacement Policy (RFR) [25], discussed in Chapter 2, can extend the life of NVMs, but they are challenging to implement as they are designed for write-optimized cache memory and do not integrate well with traditional wear-leveling schemes and NVM architecture. A more NVM-friendly replacement policy, combined with an effective wear-leveling method, can significantly enhance the lifespan of NVM LLCs. The major contributions of this chapter are as follows

- We analyse write variations in the last level NVM cache and draws meaningful conclusions.
- We propose Write Aware Last level Non-Volatile Caches (WALL-NVC), which can reduce the intra-set variation, thereby increasing its lifetime.
- For WALL-NVC, we use an NVM-friendly replacement policy called Least Recently Used Cold Block (LRU-CB), which also contributes to increase the lifetime.
- We test WALL-NVC using SPEC 2006 [30] benchmarks on the gem5 cycle-accurate simulator [26], and show that our proposed method outperforms other state-of-the-art solutions.

4.2 Motivation

To study the write variations of different applications, we analyze the maximum and average writes to the LLC in a unicore architecture. We model a unicore system

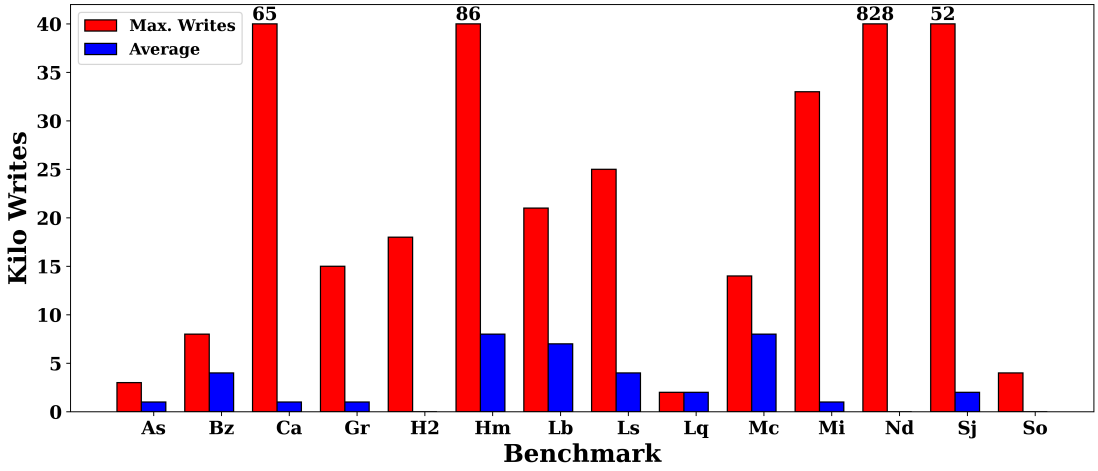

Figure 4.1: Average and maximum writes per way (Kilo writes per 1 billion instructions) of various SPEC CPU2006 benchmarks: height difference between bars of a given benchmark indicates intensity of write level variations.

in gem5 with two levels of cache and main memory. The L1-I and L1-D caches are configured as 32 KB, 2-way set associative. The unified L2 cache is 512 KB, 8-way set associative, and we use 8 GB of main memory. The block size is 64 bytes. The number of kilo writes per 1 billion instruction window for selected benchmarks from the SPEC CPU2006 suite is shown in Figure 4.1. We plot the maximum writes per way and the average writes across ways. Our study reveals that write variations can occur within and across different sets. This highlights the importance of implementing an effective wear-leveling policy for NVM-based LLCs. Popular cache replacement policies, such as Least Recently Used (LRU) and Pseudo LRU, consider the recent use of a cache block when selecting a victim block for replacement. However, in NVMs where write endurance is critical, the number of writes to the victim block can affect the cache memory's lifespan. To our knowledge, current wear-leveling techniques do not explore the impact of replacement policies on improving lifespan. EqualWrites [12], which reduces intra-set write variation, compares LRU and random replacement policies. Nevertheless, neither of these policies is designed explicitly for NVMs and do not show much impact on its lifetime. This motivates us to explore how to effectively combine a wear-leveling technique with a customized replacement policy to enhance the lifetime of NVM caches.

4.3 Proposed technique

We propose the Write Aware Last Level Non-Volatile Cache (WALL-NVC) to enhance the lifespan of NVMs in applications with non-uniform writes. WALL-NVC also protects against targeted malicious attacks through repeated writes to specific blocks. Unlike most state-of-the-art wear-leveling techniques, WALL-NVC utilizes a dual-stage wear-leveling approach. The first stage introduces a new Least Recently Used Cold Block (LRU-CB) replacement policy, which optimizes the selection of victim blocks for cache replacement in NVMs. The second stage applies a traditional write distribution strategy that works with LRU-CB to extend the memory's lifespan. The following sections will delve into the details of these stages.

4.3.1 Least Recently Used Cold Block (LRU-CB)

An effective cache replacement strategy for NVMs should enhance write endurance and minimize intra-set write variation. It should ideally prioritize the retention of the most frequently accessed blocks, preventing frequent evictions and reducing write variation among blocks. In scenarios where the cache hit rate is high, the number of replacements is low, thus minimizing the impact of replacement policies. Conventional policies like LRU and Pseudo LRU do not account for the write count of a block in their victim block selection. To address these concerns, we propose a simple, NVM-friendly cache block replacement policy known as the Least Recently Used Cold Block (LRU-CB). Figure 4.2 shows the victim block selection using LRU-CB policy in Set A of an 8-way set associative cache. The fundamental idea behind the LRU-CB policy is to designate a block with fewer write occurrences in the cache set as the victim block, thereby promoting a more uniform distribution of writes within the set. To ensure infrequent eviction of blocks, we calculate a weighted aggregate average of each block's LRU age and write index. The block with the lowest aggregate average is chosen as the victim block. To facilitate this, a write counter is associated with each block. When the write counter of a set reaches its saturation value, the counters for all blocks in that set are bitwise right-shifted. This downgrading of the counter value leads to the loss of the least significant bit, which introduces a minor

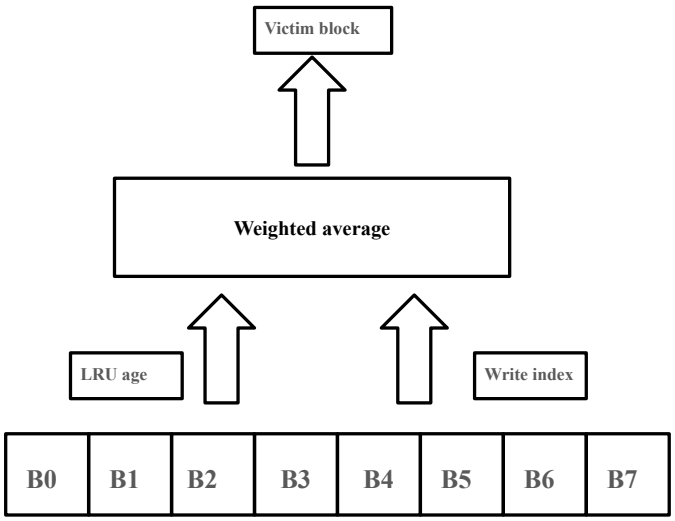

Figure 4.2: Schematic representation of victim block selection using LRU-CB policy in Set A of an 8-way set associative cache

loss of precision. This mechanism ensures that the write counters are handled before reaching saturation, but at same time reliable history is preserved. We evaluate the impact of LRU-CB using various benchmarks and observe that it enhances the lifespan of NVM caches. In our experiments we use a weight of 80% for LRU age and 20% for write count index for selecting victim block in LRU-CB policy. The details of analysis on different weights for LRU age and write index is discussed in Section 4.4.4. Replacement policies are active only in the event of cache replacement. Due to this LRU-CB has limited impact on the lifetime improvement of NVM LLC. This modest improvement brings out the need for an additional augmenting technique to further enhance the performance of LRU-CB.

4.3.2 Impact of LRU-CB with Write Distribution

Write-aware replacement policies exhibit limited influence on the endurance of NVM caches when handling applications with high L1 cache hit rates due to fewer triggered evictions. Write distribution policies contribute to enhanced lifetime by maintaining an even distribution of writes. For extending the lifetime of NVM caches, combining a robust wear-leveling policy with a write-aware replacement policy, as opposed to

employing them separately will be a better approach.

To assess the impact of LRU-CB in conjunction with a standard state-of-the-art wear-leveling technique, we compare the effectiveness of the EqualWrites [12] technique with the pseudo LRU policy and LRU-CB. Figures 4.3, 4.4, and 4.5 illustrate the relative lifetime, intra-set variation, and hit rate of NVM caches, respectively, across different benchmarks in the SPEC CPU2006 suite. The graphs reveal that EqualWrites with LRU-CB enhances the lifetime of the L2 cache by up to 1.39x compared to the combination of EqualWrites with pseudo LRU. LRU-CB reduces intra-set variation by up to 83.08% without impacting the hit rate. This improvement is consistent across all benchmarks, affirming that LRU-CB stands out as a superior cache block replacement algorithm for NVM caches.

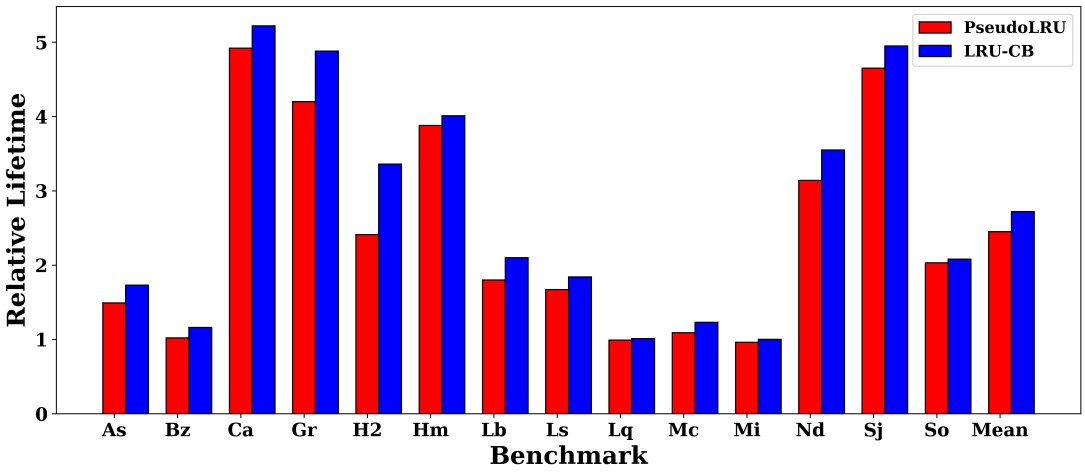

Figure 4.3: Comparison of relative lifetime of NVM based L2 cache using EqualWrites with Pseudo LRU and LRU-CB replacement policies.

4.3.3 Write Distribution in WALL-NVC

The LRU-CB policy enhances the performance of the EqualWrites technique; however, this improvement comes at a significant cost. This is because LRU-CB requires additional counters beyond those utilized by EqualWrites. The counters used in LRU-CB results in 1.17% storage overhead and the EqualWrites adds another 2% as storage overhead. This prompted us to explore the implementation of a wear-leveling

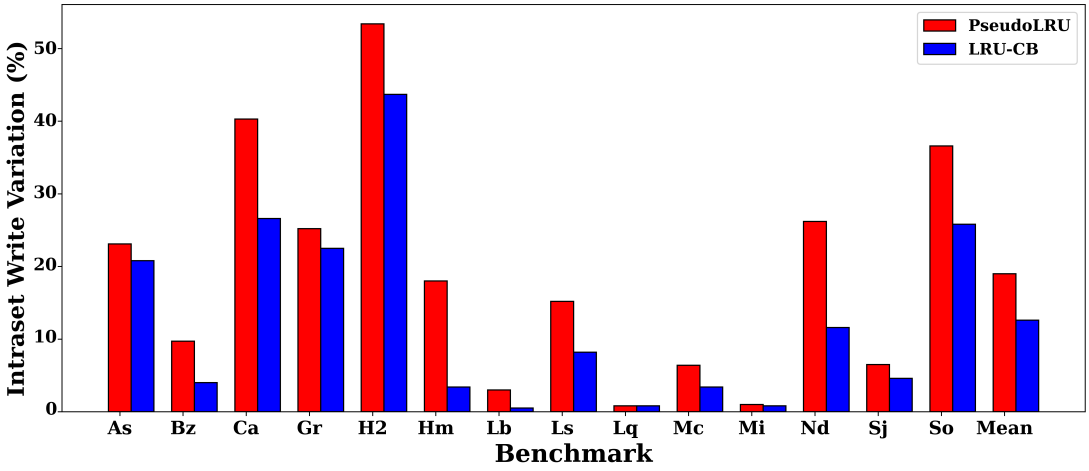

Figure 4.4: Comparison of intra-set variation of NVM based L2 cache using Equal-Writes with Pseudo LRU and LRU-CB replacement policies.

policy that mitigates intra-set variation, enhances lifetime, and synergizes with LRU-CB. Cache memory blocks which are frequently accessed for write operation are termed as hot blocks, whereas blocks which are rarely accessed for write operation are called as cold blocks. Like other popular wear-leveling techniques, WALL-NVC operates on redirecting writes from hot blocks to cold blocks.

In an n-way set associative WALL-NVC, each set is equipped with $(n + 1)$ counters: one set counter and n block counters. When a write hit occurs in WALL-NVC, the corresponding set and block counters are updated. Once the set counter surpasses a predetermined threshold T, it identifies a write redirection target among the blocks in that set. The block with the least writes is favoured as the redirection target, typically with zero write count. If such a block is available, a swap is initiated between the accessed block and the chosen redirection target. In cases where the target block is invalid, the data is written to the target block instead of swapping, and the hot-line is invalidated. If a block with zero write count is not available and the target is not found, all counters, including the set counter, are decremented by the value of the least written block. This delay in write redirection is implemented to avoid unnecessary redirection when the write pattern to the set is more uniform. It is important to note that reducing the block counter value does not impact its functionality. Furthermore, the reduction in counter values delays the need for bitwise

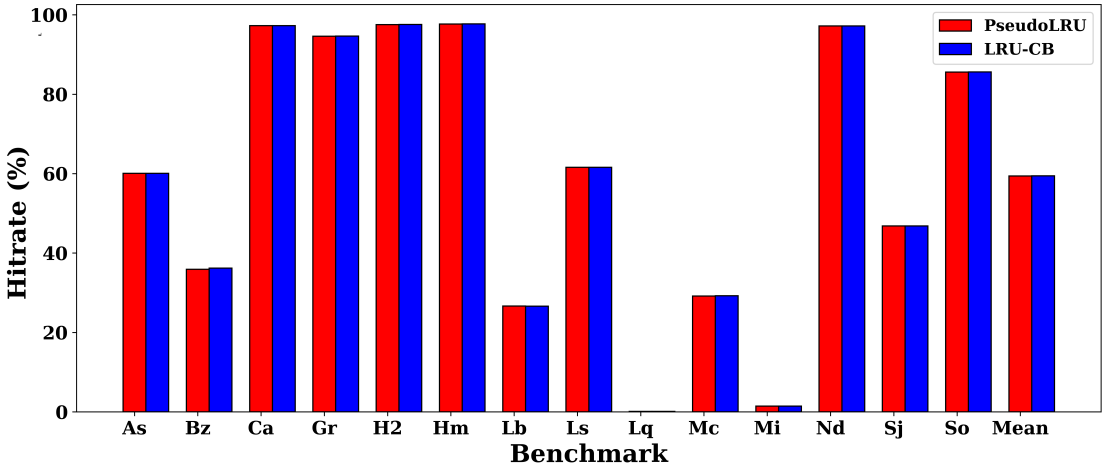

Figure 4.5: Comparison of hit rate of NVM based L2 cache using EqualWrites with Pseudo LRU and LRU-CB replacement policies.

right shift operations of counters for replacement victim selection, thereby enhancing the precision of the technique.

To comprehend the working of WALL-NVC, we use an illustration featuring a four-way set associative cache block within WALL-NVC, characterized by a threshold value of 50. Each cache set is linked to a set counter and four block counters to monitor the write count of each set and block, respectively. Let us focus on a specific set, denoted as A, with four blocks labeled $B0, B1, B2$, and $B3$. Assume A's set counter and block counters are represented in the initial row of Figure 4.6. A write hit in a block increments both the block counter and the set counter of A. Upon reaching the threshold value (50), the set counter looks for a write redirection target for the heavily written block ($B2$). If no target block with zero counts is found, all block counters and the set counter values are reduced by the least count value creating a block with zero count. In this case, 2 for $B3$ is reduced from all the block counters and set counters, making the write count of the block $B3$, and therefore, it is a write redirection target. After the decrement operation, the cache operates normally by incrementing the counters on subsequent write hits. Upon reaching the threshold again and initiating write redirection, the redirection occurs by swapping the contents of the most written block ($B2$) with the least written block ($B3$) using a swap module. Since $B2$ and $B3$ are valid blocks, the write redirection results in an additional write

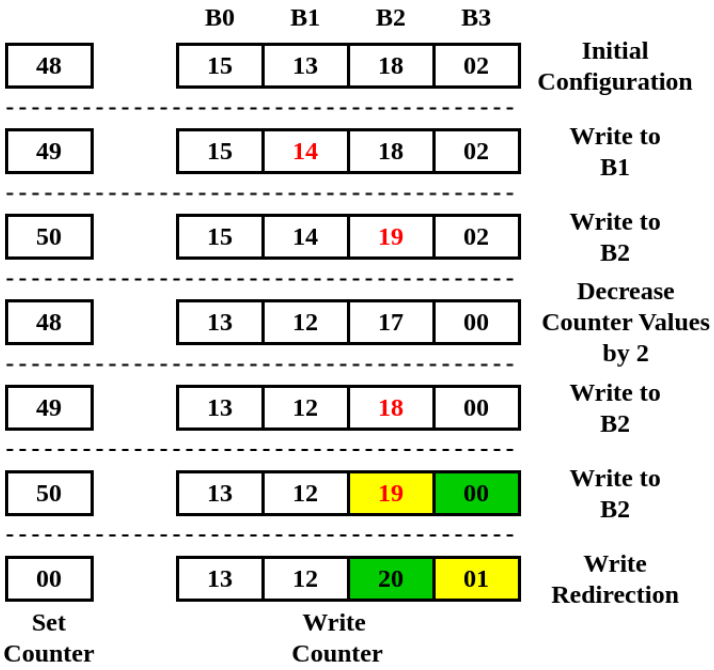

Figure 4.6: Sample counter updating of WALL-NVC for threshold value, T=50

in both $B2$ and $B3$, resetting the set counter. The block counters remain the same as the value of the block counter, which indicates the write frequency of the blocks. By preserving block counter values, WALL-NVC ensures that hot blocks are always placed in locations with less write access and cold blocks are placed in locations with more write access.

4.4 Experimental Setup and Result Analysis

The benchmarks are executed on an un-optimized NVM LLC (baseline), two state-of-the-art wear-leveling methods (EqualWrites and EqualChance), and the newly introduced WALL-NVC, utilizing a threshold value T=50 (WALL-NVC50). We analyzed the lifetime improvement for our proposed system for various values of threshold T, from $T = 10$ to $T = 50$. Our analysis shows that increasing the size threshold will not improve the gains compared to the overhead associated with it. Considering the gains and counter overheads, we fix the threshold value as 50 for our experiments. Specific configuration details of the system can be found in Table 4.1.

Table 4.1: System Configuration

CPU	2 GHz, Unicore, Dual-Core, Quad- Core,
L1 Cache	Private, 32 KB, SRAM based split cache, 64 B block, 4-way set associative
L2 Cache	Shared 512 KB | 2MB, NVM based unified cache 64 B block, 8-way set associative
Main Memory	8 GB

4.4.1 Impact on Write Variation

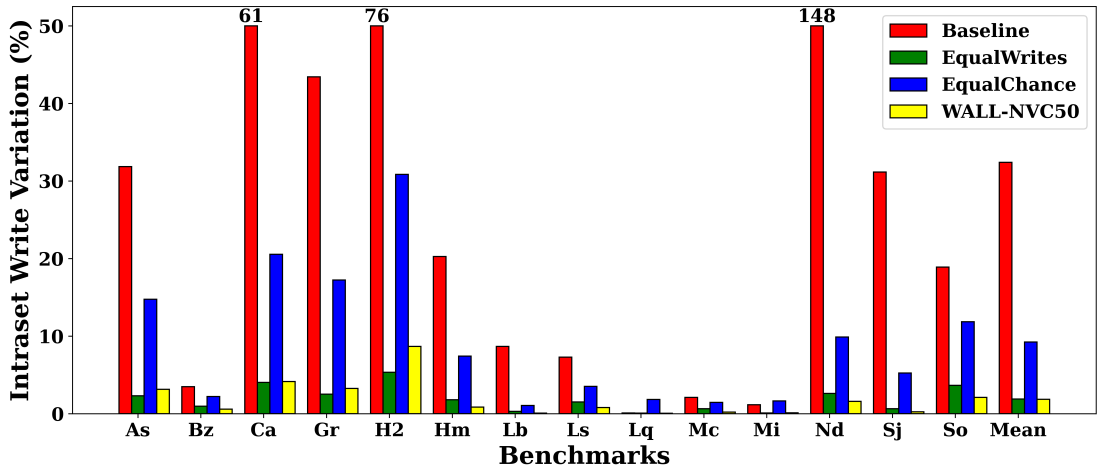

Figure 4.7: Comparison of $IntraV$ for various NVM architectures in unicore system of L2 cache size 512KB

Figure 4.7 compares the intra-set variation ($IntraV$) among various cache architectures proposed above in unicore systems. The analysis reveals a fair distribution of writes across different set ways exists in benchmarks such as *leslie3d*, *lbm*, *mcf*, *milc*, and *bzip2* across all architectures , resulting in lower $IntraV$. This tendency is particularly pronounced in benchmarks with mid and high WPKI. Notably, low WPKI benchmarks like *namd*, *calculix*, and *gromacs* exhibit significant enhancements in $IntraV$. This leads us to the conclusion that write variance is more dependent on the pattern of write hits within a given set than the total number of writes over time. Additionally, regardless of the benchmark classification, WALL-NVC50 consistently

demonstrates low write variance, making it a suitable addition to NVM caches.

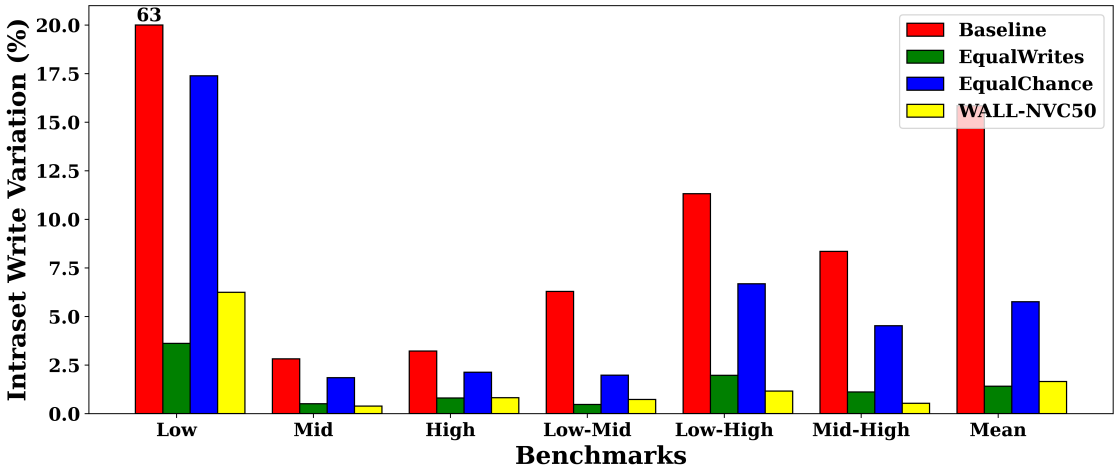

Figure 4.8: Comparison of *IntraV* for various NVM architectures in dual-core system of L2 cache size 512KB

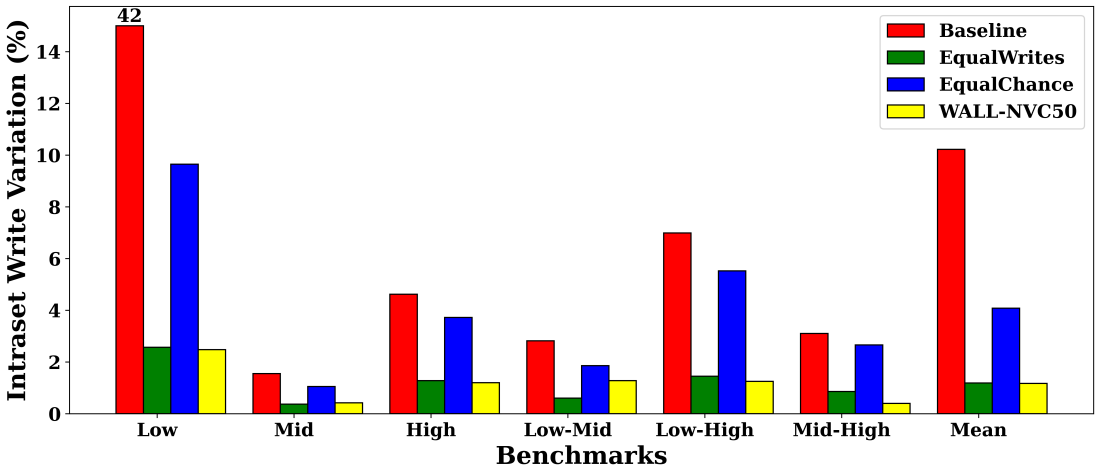

Figure 4.9: Comparison of *IntraV* for various NVM architectures in quad-core system L2 cache size 512KB

Intra-set variation in dual-core and quad-core systems is plotted in Figure 4.8 and Figure 4.9, respectively. Since the multi-core framework requires multiple benchmarks, workload composition is created based on WPKI values, categorized as Low, Mid, Low-High, Mid-High, etc. Analysis reveals that shared NVM LLC is accessed by multiple applications in multi-core systems, nullifying the write variations created by

one core with natural random balancing from other cores. Consequently, only minor improvements in lifetime are achieved using Mid-High workloads in dual-core and quad-core systems.

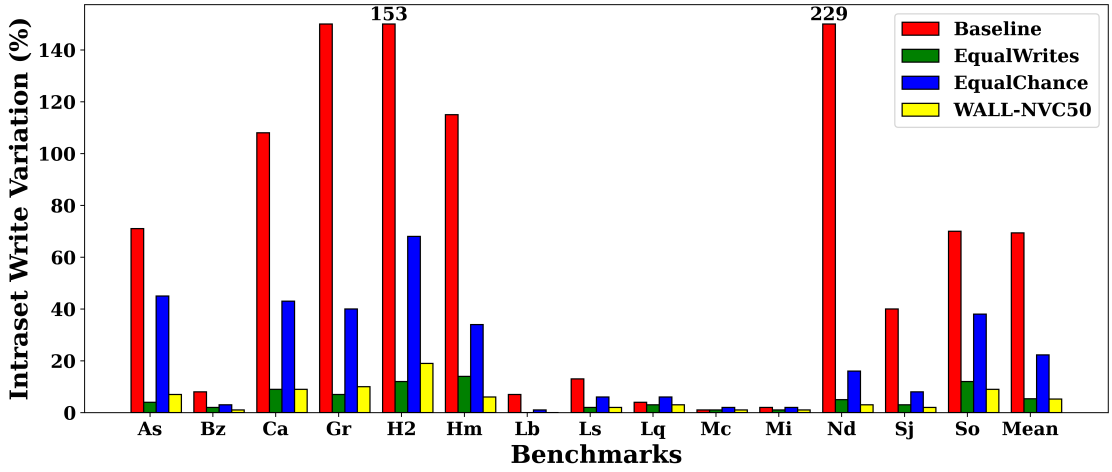

Figure 4.10: Comparison of *IntraV* for various NVM architectures in unicore system of L2 cache size 2MB

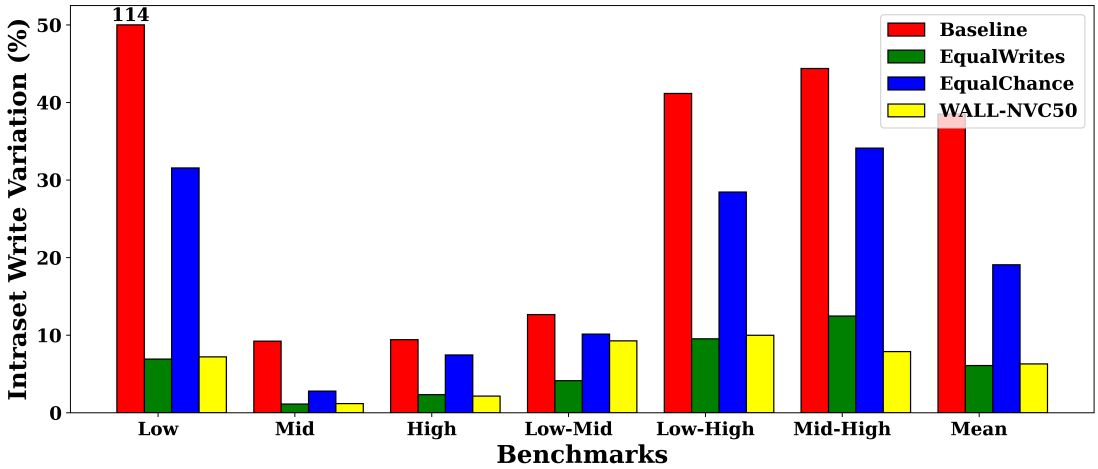

Figure 4.11: Comparison of *IntraV* for various NVM architectures in dual-core system L2 cache size 2MB

To study the impact of WALL-NVC on larger caches, we ran our experiments on an L2 cache size of 2MB. Like 512KB, we run experiments on unicore, dual-core and quad-core systems. We also use the same benchmark mixes used for dual and

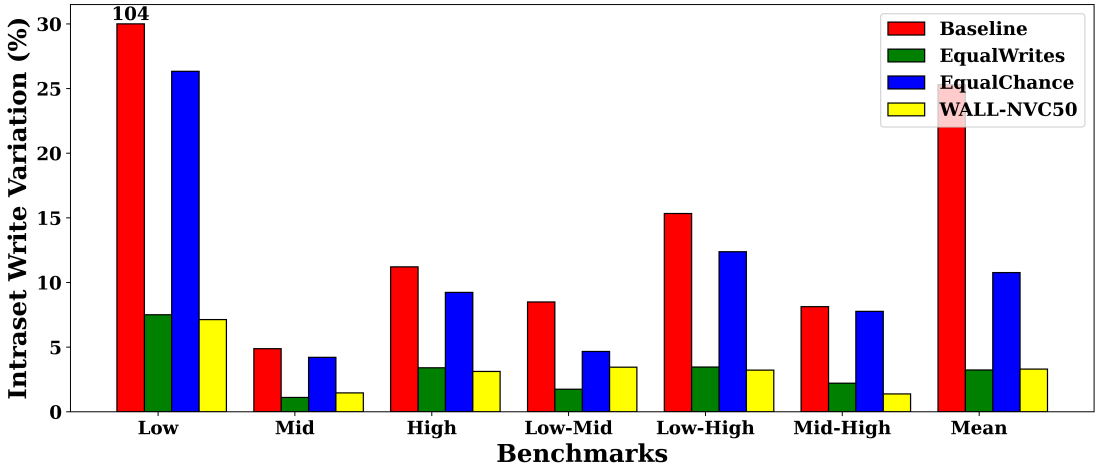

Figure 4.12: Comparison of *IntraV* for various NVM architectures in quad-core system L2 cache size 2MB

quad-core systems for 512KB cache size for 2MB. Figures 4.10, 4.11, 4.12 show the impact of various wear-leveling techniques and the baseline a 2MB NVM L2 cache for unicore, dual-core and quad-core respectively. The results follow a similar trend to 512KB except for the increased intra-variation in the 2MB cache. As cache size increases for a given application execution window, intra-set write variation increases. For a given application execution window, increasing cache capacity improves the hit rate and increases repeated access to a few cache blocks. This increases the intra-set write variation, as evident from the IntraV value of 2MB. Hence, the write distribution mechanism of WALL-NVC has a better impact on the larger LLCs as the frequency of write redirection is high, reducing the intra-set write variation.

4.4.2 Impact on Relative Lifetime

Relative lifetime is a key indicator of the effectiveness of the wear-leveling techniques implemented in different memory hierarchies. Most state-of-the-art lifetime enhancement techniques use relative lifetime for analysis as they quantify the impact of the proposed architecture on the maximum number of writes recorded in the cache memory with respect to the baseline architecture.

Figure 4.13 compares relative lifetimes across various cache architectures of

512KB size in unicore systems. The experimental results show that WALL-NVC50 significantly improves the lifetime for applications like *calculix, gromacs, h2ref, hmmer and namd*. These applications have a higher L2 cache hit rate and intra-set write variation than others, as the writes are concentrated in a few memory locations. As write variation is high on these applications, the WALL-NVC50 can distribute the write concentrated on a few blocks to other less written blocks, reducing maximum writes and write variation. Hence, the impact of WALL-NVC is prominent in these benchmarks. As anticipated, benchmarks with high WPKI exhibit limited improvement in lifetime due to intense writes to LLC. Also, we observe that the benchmarks with high cache hit rates show better lifetime time with WALL-NVC50. On average, WALL-NVC50 enhances lifetime by 2.90x compared to the baseline architecture, showing 1.16x and 1.18x improvements over EqualWrites and EqualChance, respectively. The evaluation extends to dual-core and quad-core

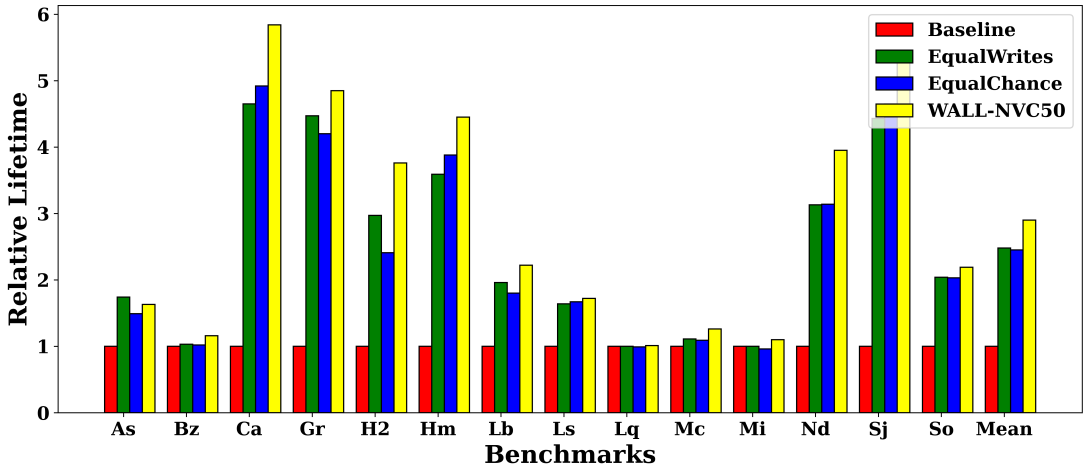

Figure 4.13: Comparison of relative lifetime for various NVM architectures in unicore system of L2 cache size 512KB

systems, where WALL-NVC50 shows an average lifetime improvement of NVM by 2.25x and 1.63x compared to baseline systems, as depicted in Figure 4.14 and Figure 4.15. WALL-NVC50 shows better lifetime improvement for dual-core systems for low, mid and low-high category benchmark combinations. Like unicore systems, these combinations have better cache hit rates and high intra-set write variation; hence, the impact of WALL-NVC50 is prominent.

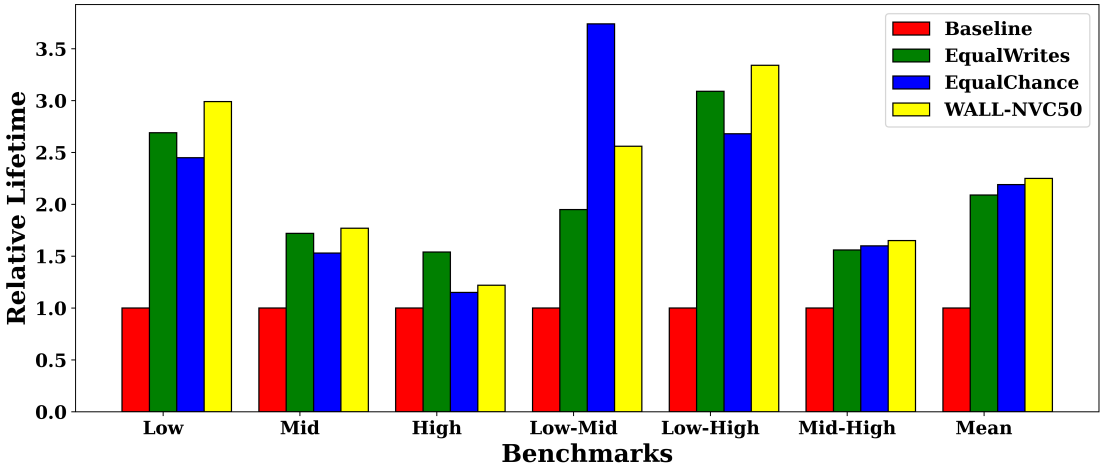

Figure 4.14: Comparison of relative lifetime for various NVM architectures in dual-core system L2 cache size 512KB

As we move from dual-core to quad-core, the impact of WALL-NVC decreases as the write distribution pattern becomes more uniform due to multi-core architecture cache coherence. Compared to EqualWrites, it improves by 1.07x on dual-core systems and 1.02x on EqualChance. For quad-core systems, lifetime improvements of 1.10x and 1.02x are achieved, respectively.

We analyse the effectiveness of the proposed technique on relative lifetime with respect to varying cache size by experiment with cache size of 2MB also. Figures 4.16, 4.17, 4.18 shows the relative lifetime improvement values for cache size 2MB for unicore, dual-core and quad-core respectively. The results follow similar trend as for intra-set variation. As mentioned before the as cache size increases the intra-set write variation increases and the impact of WALL-NVC50 also becomes significant.

4.4.3 Impact on IPC

WALL-NVC uses the LRU-CB replacement policy, which has little impact on the cache hit rate. The wear-leveling algorithm does not invalidate the cold blocks. Instead, it keeps the block within the cache memory; hence, there is no impact on the cache hit rate. Since there is no change in hit rate, there is no change memory access time and hence WALL-NVC does not impact the system's instructions per

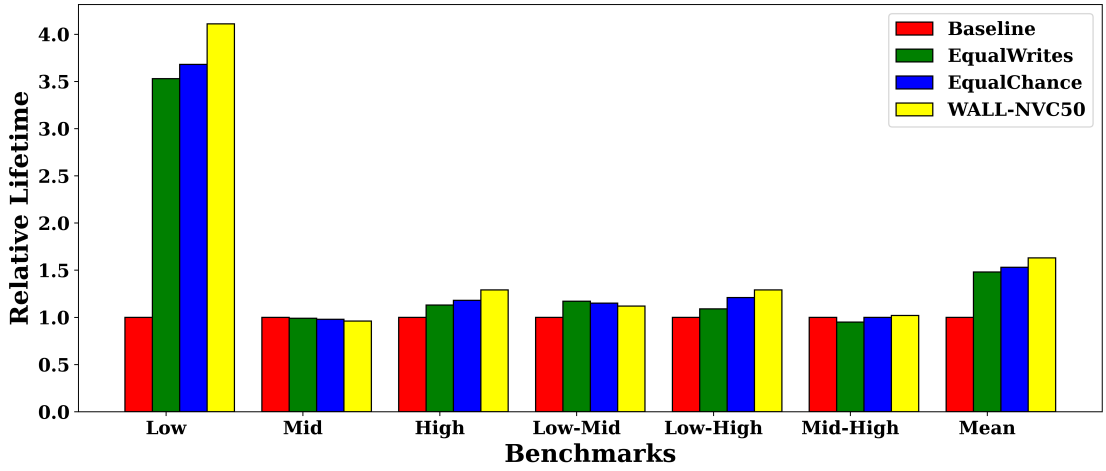

Figure 4.15: Comparison of relative lifetime for various NVM architectures in quad-core system L2 cache size 512KB

Table 4.2: Relative lifetime improvement (LT) and $IntraV$ of WALL-NVC for different threshold values

	Unicore		Dual-core		Quad-core	
	LT	$IntraV$	LT	$IntraV$	LT	$IntraV$
Baseline	1	32.41	1	15.85	1	10.22
10	2.32	6.55	2.08	7.28	1.31	6.58
30	2.54	2.46	2.63	1.48	1.68	0.34
50	2.90	1.85	2.25	1.65	1.63	1.17
70	2.56	4.08	2.23	2.59	1.59	1.43
100	2.57	4.20	2.35	2.26	1.53	1.52

cycle (IPC).

4.4.4 Sensitivity Analysis

We study the influence of the threshold value (T) through experimentation with five distinct values, $T = 10, 30, 50, 70, 100$. After evaluating lifetime improvement and its corresponding overhead, we converge to a default value of T as 50. Additionally, we perform an extensive sensitivity analysis on various threshold values. Table 4.2 displays the relative lifetime and intra-set write variation, utilizing different threshold values in WALL-NVC. The performance of the proposed architecture can be influenced

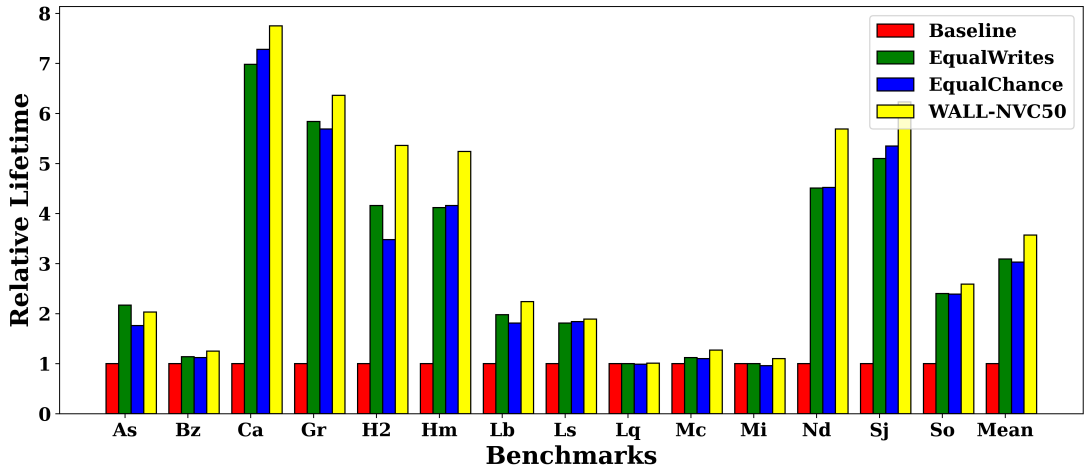

Figure 4.16: Comparison of relative lifetime for various NVM architectures in unicore system of L2 cache size 2MB

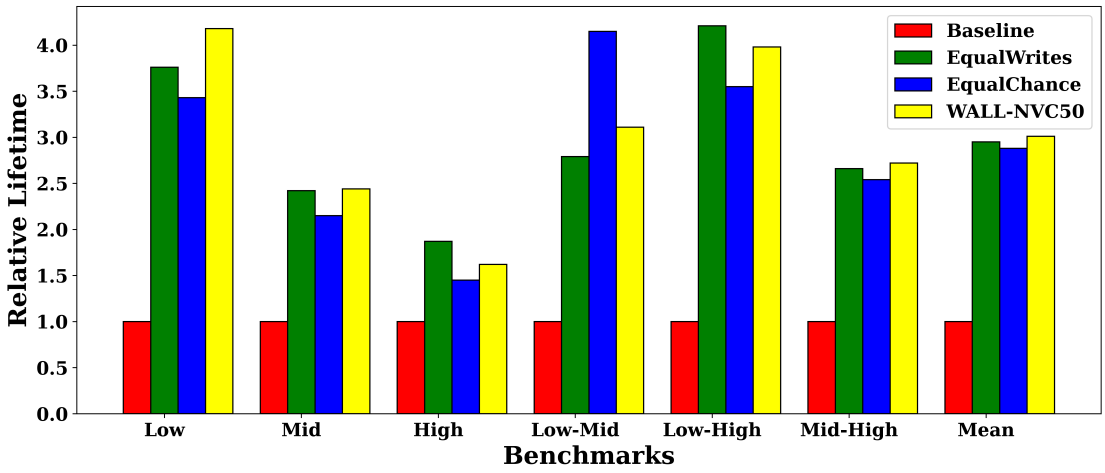

Figure 4.17: Comparison of relative lifetime for various NVM architectures in dual-core system L2 cache size 2MB

by the weight assigned to LRU-CB during the selection of a victim block for cache replacement. As discussed previously, we calculate the weighted aggregate average of each cache block by considering both its LRU age and write count index. Two variants are explored: (a) 80% for LRU age and 20% for write count index (0.2W), and (b) 60% for LRU age and 40% for write count index (0.4W). These variants are compared in a unicore system, with the results for $IntraV$, relative lifetime with respect to the

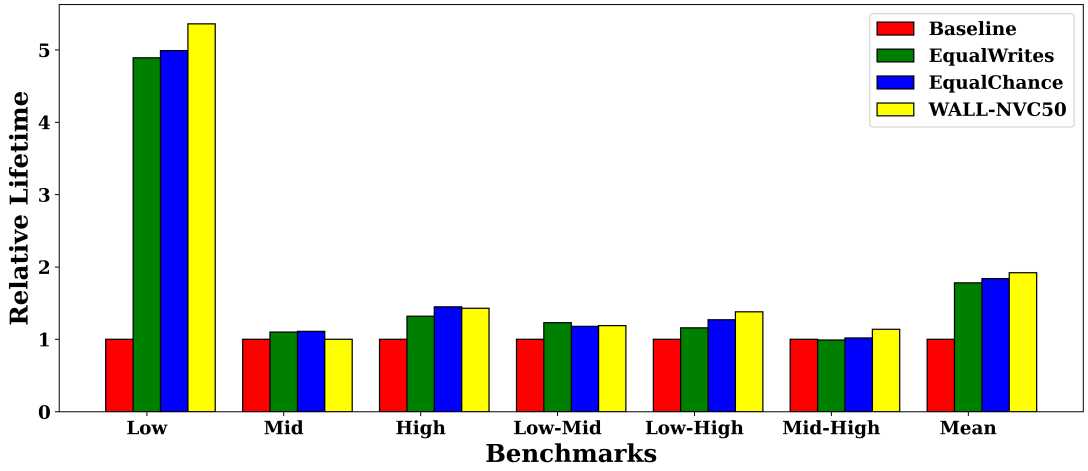

Figure 4.18: Comparison of relative lifetime for various NVM architectures in quad-core system

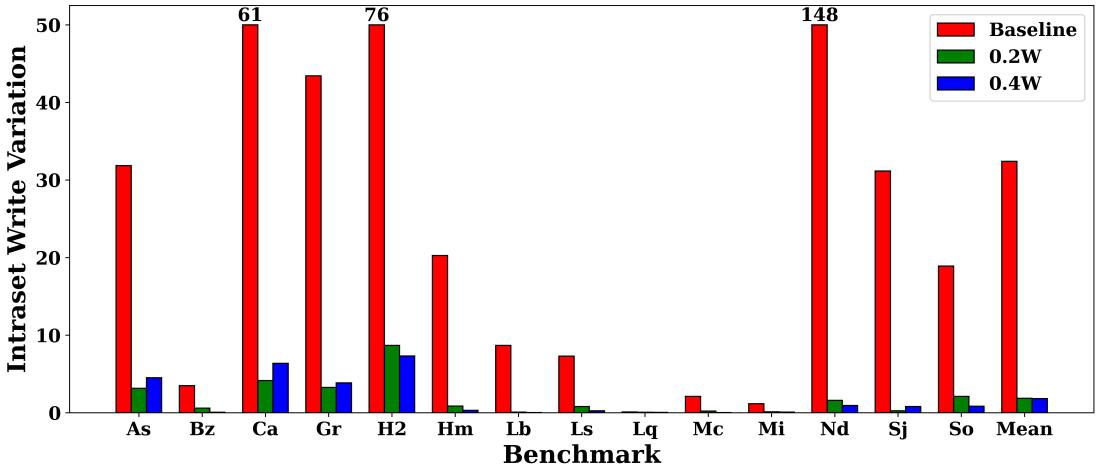

Figure 4.19: Comparison of *IntraV* for WALL-NVC variants in unicore system

baseline, and LLC hit rate presented in Figures 4.19, 4.20, and 4.21, respectively. We see that 0.2W yields a 1.13x improvement in lifetime and a 2.16% enhancement in *IntraV* compared to 0.4W. Interestingly, there is no impact on the hit rate across various benchmarks for these two variants. Considering that both 0.2W and 0.4W outperform the baseline, it is suggested that a minimum weightage to LRU-CB (0.2W) should be assigned to obtain a suitable victim block. We also see that an excessive emphasis on the write count index (0.4W) diminishes the significance of LRU age.

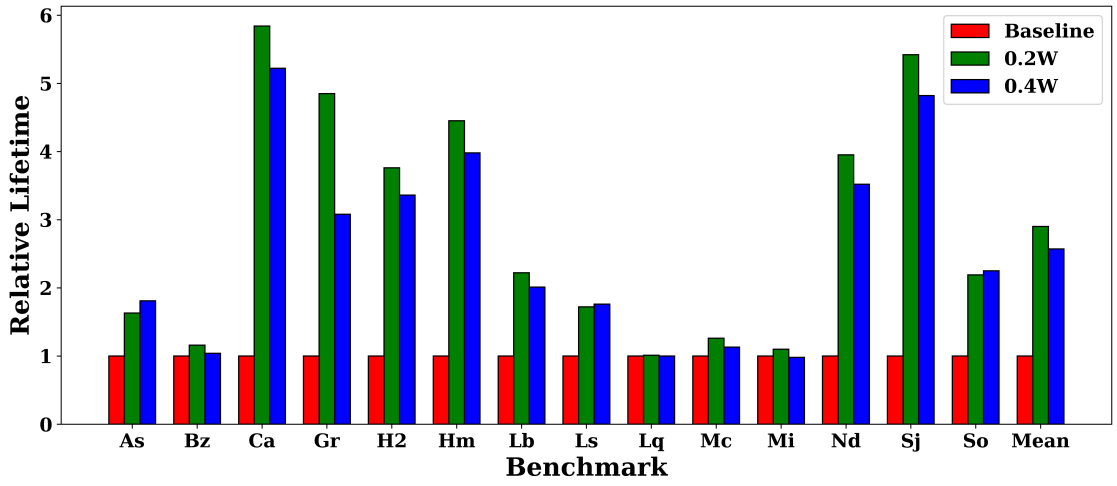

Figure 4.20: Comparison of relative lifetime for WALL-NVC variants in unicore system

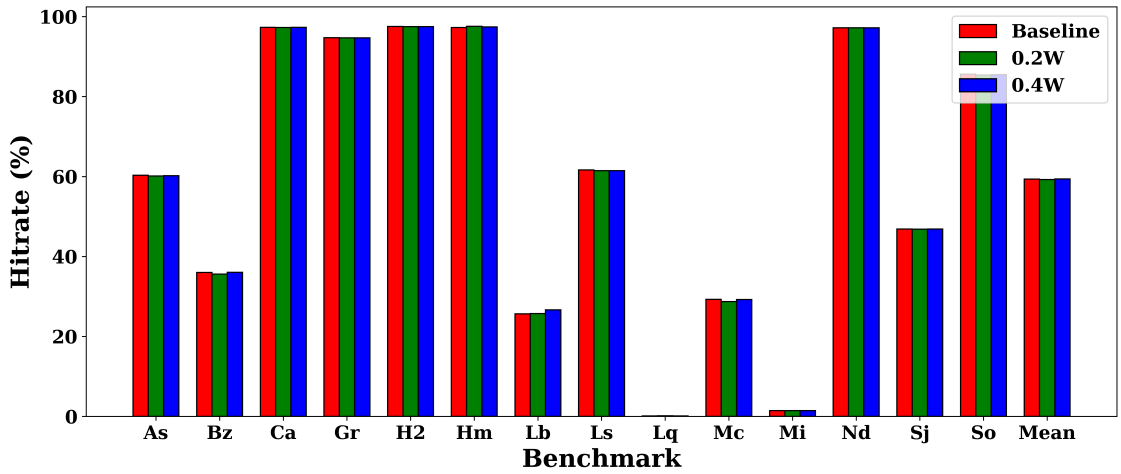

Figure 4.21: Comparison of LLC hit rate for WALL-NVC variants in unicore system

4.4.5 Overhead Analysis

WALL-NVC utilizes two types of counters: a set counter assigned to each set and a block counter assigned to each block. A swapping module is also required to interchange the contents of hot and cold data blocks. This swapping module consists of 64 buffers, each with a size of 64 bytes, resulting in a total storage overhead of 2% with respect to the baseline architecture. Based on the analysis using CACTI tool

[35], the SRAM-based counters and swap buffers result in a maximum power and area overhead of 0.47% and 1.47%, respectively, compared to the baseline configuration. The cache block replacement policy, LRU-CB, utilizes the same counters for victim selection, thereby incurring no additional overhead.

4.5 Conclusion

The limited write endurance of NVM presents a critical challenge. In this study, we introduce a novel architecture named WALL-NVC, which employs a write distribution policy and an NVM-friendly, LRU-CB cache replacement strategy to enhance the lifetime of NVM caches. Our observations indicate that the write-distribution and write-aware replacement policies play equally significant roles in performance improvement. Experimental results demonstrate that our approach enhances the longevity of NVM caches for unicore, dual-core, and quad-core systems with minimal area and power overhead.

CHAPTER

5

Virtually Split Last Level Non-Volatile Cache

This chapter discusses Virtually Split Cache (ViSC), an innovative wear-leveling technique, and its variants, which enhance the lifetime of SLC NVM LLCs. ViSC exploits the write variation between data and instruction blocks to ensure that all LLC cache blocks are accessed for writes at a more uniform rate. The proposed technique shows significant improvement in lifetime compared to other state-of-the-art techniques. ViSC also has less overhead.

5.1 Introduction

In the previous chapter, we discussed WALL-NVC, a technique for improving the lifetime of SLC NVM by customizing the replacement policy and interchanging the frequently written blocks and less written ones to ensure uniform write distribution. This reactive approach of triggering the wear-leveling technique using counters and thresholds results in storage overhead. To address this issue, we propose a proactive wear-leveling technique, Virtually Split Cache (ViSC) and its variants. The following

sections discuss ViSC and its variants, E-ViSC and P-ViSC. The key idea behind ViSC is to logically split the cache sets of a unified cache and reserve them for instruction and data blocks. Based on a prefixed interval, these blocks are remapped; blocks previously reserved for data (hot blocks) will be reserved for instruction (cold) blocks and vice versa. Unlike WALL-NVC and other state-of-the-art techniques [11] [12] [13] [14], ViSC does not extensively use counters; hence, the associated area and power overhead are less. The key contributions of this chapter are as follows

- We study the write variation among instruction and data blocks of different applications in LLCs and its impact on lifetime.
- We propose a wear-leveling technique and its variants that can reduce the write variation in NVM LLCs and improve their lifetime. We model the proposed techniques on gem5 [26] and compare them in terms of performance metrics using SPEC CPU2006 benchmarks [30].

5.2 Motivation

Traditional wear-leveling mechanisms in NVMs use write counters to identify hot and cold data blocks, which incurs higher overhead due to the number of counters and associated circuitry. The high overhead of wear-leveling techniques motivated us to explore the possibility of a low-overhead wear-leveling technique.

Traditional architectures use split L1 cache architecture, segregating instructions and data, while L2 caches typically adopt a unified approach, combining both instructions and data. Data blocks undergo more write operations compared to instruction blocks [15]. To study the memory access pattern of L2 caches, we run various benchmark applications on the gem5 simulator.

Figure 5.1 shows the memory access pattern of in a unified L2 cache of 512KB capacity while running various SPEC CPU2006 benchmark applications for one billion instructions . The results indicate that the L2 cache accesses are predominantly data blocks. The experimental result also reinforces that data blocks are frequently accessed (hot blocks), whereas instruction blocks are less frequently accessed (cold blocks). The scheme of categorizing cache blocks into instruction and data ways can

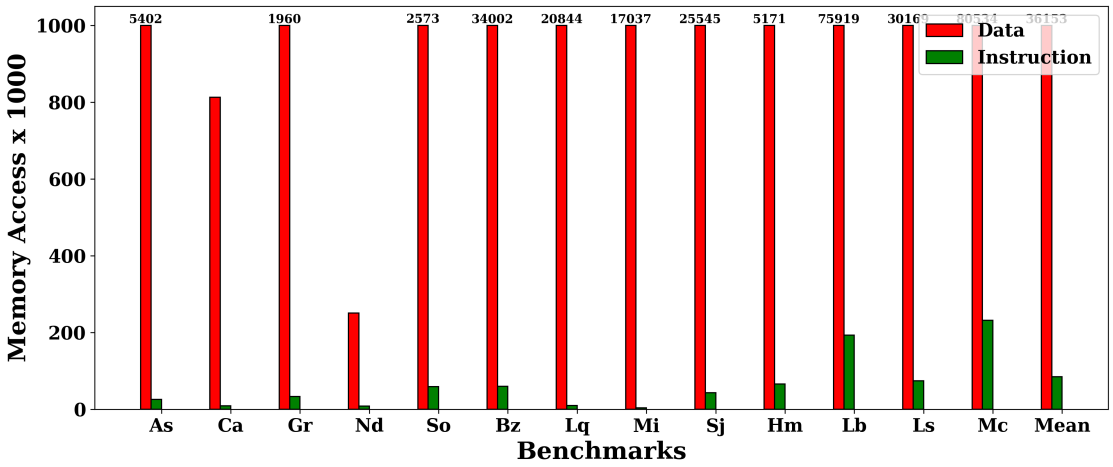

Figure 5.1: L2 cache access pattern for different benchmark applications

be used to identify hot and cold blocks for wear-leveling. This is the the rationale behind our proposed techniques and can be implemented with minimal overhead in NVM caches. Coming sections discuss the proposed technique in detail.

5.3 Proposed Technique - ViSC

Virtually Split Cache (ViSC) is a novel low-overhead technique for improving the lifetime of SLC NVM unified L2 caches which logically splits them into instruction and data caches. As mentioned before, popular NVM wear-leveling strategies employ counters at various levels of memory hierarchy, resulting in high overhead. Instead of finding hot and cold data blocks using counters to distribute the writes, ViSC logically splits the cache sets into instruction and data ways to reserve them for storing respective blocks only. This mapping is periodically changed to ensure that every block will be a hot block (when reserved for data) for some time interval and a cold block (when reserved for instruction), ensuring better write distribution. This approach reduces the counter overheads required for hot and cold block identification. We propose three variants of ViSC and details are given below.

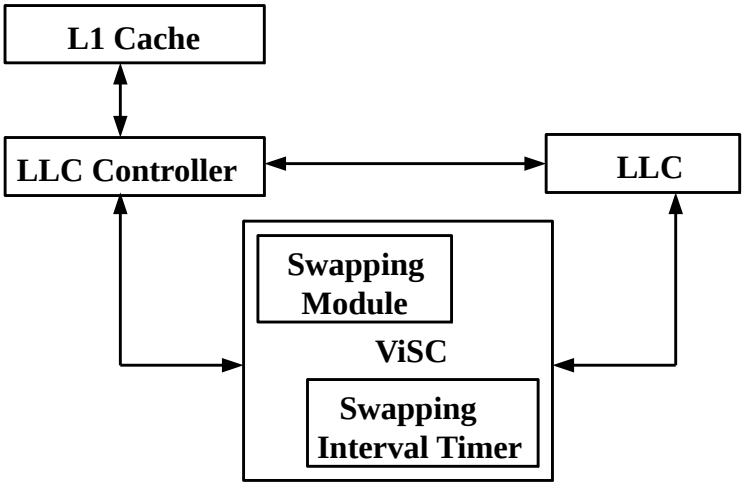

Figure 5.2: L1 cache - LLC interaction through the proposed ViSC module

5.3.1 Virtually Split Cache (ViSC)

ViSC partitions the unified NVM LLC into distinct data and instruction ways, dedicating each for storing the respective blocks. ViSC periodically reorganizes this logical mapping at fixed intervals, transitioning heavily utilized data ways to function as instruction ways and vice versa. This process ensures a balanced distribution of heavily accessed ways throughout the cache memory. The proposed architecture integrates the ViSC module with the LLC controller as shown in Figure 5.2. Each L1 cache miss reaching the LLC controller is directed to the ViSC module for necessary background verification and updating. The ViSC module incorporates a swapping interval timer, facilitating the mapping of instruction ways and data ways. This process, managed by the swapping module, involves signalling to designate the current k-ways for instruction storage as data ways while also assigning new k-ways for instruction storage. This reassignment and interchange of ways between instruction and data is called set reorganization. The swapping module initiates set reorganization at regular intervals using an appropriate timer. Upon receiving a write request to the LLC, it verifies if the time elapsed since the last set reorganization exceeds a predetermined threshold. If it does, the swapping module is activated. By converting heavily written data ways (hot ways) to less utilized instruction ways (cold

ways), the number of writes is reduced, thereby minimizing write variation.

Algorithm 1 provides a comprehensive outline of the operations executed within the ViSC module. The line numbers 1 - 8 of Algorithm 1 defines important parameters such as cache associativity, number of instruction ways, etc. Initially, instr_start is set to 0, and data_start is set to 3. Since the associativity is set to 8, way 0, 1 and 2 are reserved for instruction blocks and way 3 to way 7 is reserved for data and are stored in instruction_way_list and data_way_list, respectively. The algorithm processes each cache request (R, line number 9) sequentially. For read requests, it performs the usual read operation (normal read operation, line number 11). For write requests, the algorithm checks if the elapsed time (tp) exceeds the threshold (tp $>$ threshold, line number 14). If this condition is true, a reorganization occurs, and the instruction and data ways are swapped using the swap function (line number 14). The starting indices of the instruction and data ways are updated using modular arithmetic (instr_start and data_start, line numbers 15–16) to ensure the indices wrap around the cache associativity. The updated indices are used to recalculate instruction_way_list and data_way_list (line numbers 17–18), and the algorithm proceeds with a normal write operation (line 19). If the threshold is not exceeded, the write operation proceeds without any reorganization (lines 22–23).

The swap function (line 25) exchanges the roles of the instruction and data ways by interchanging their corresponding entries in the instruction_way_list and data_way_list (lines 29–31). This reorganization also resets the timer (tp = 0, line 32), marking the beginning of a new monitoring period.

L1 caches typically employ a split organization to mitigate structural hazards within the instruction pipeline when accessing memory for instructions and data simultaneously in a single clock cycle. Conversely, higher-level caches often adopt a unified organization, accommodating instruction and data within the same cache. Here, we delve into the specifics of our proposed architecture. Let us consider an 8-way set associative L2 cache as shown in Figure 5.3, wherein we virtually allocate three ways for instructions (k=3) and the remaining five for data (8-3=5). Unlike L1 split caches, which commonly use equal-sized caches for instructions and data, our ViSC architecture may feature unequal partitioning for the L2 cache. This discrepancy

ALGORITHM 1: Operational steps in ViSC module

```
1  instr_start = 0 \\first way of instruction ways;
2  data_start = 3 \\first way of data ways;
3  cache_assoc = 8 \\cache associativity;
4  num_inst_ways =3 \\number of instruction ways;
5  t_p: time elapsed since last set reorganisation, increments every clock cycle;
6  threshold = 100000;
7  List instruction_way_list : List of ways reserved for instructions sequentially from
     instr_start. Size = 3;
8  List data_way_list : List of ways reserved for data; sequentially from data_start. Size
     = 5;
9  repeatfor every L2 cache request R and block B do
10 |  if R==read then
11 |  |  normal read operation;
12 |  else
13 |  |  if t_p > threshold then
14 |  |  |  swap(instruction_way_list,data_way_list);
15 |  |  |  instr_start = (instr_start+num_inst_ways) % cache_assoc;
16 |  |  |  data_start = (data_start+num_inst_ways) % cache_assoc;
17 |  |  |  instruction_way_list = {instr_start, instr_start+1, instr_start+2 };
18 |  |  |  data_way_list = {data_start, data_start+1, data_start+2, data_start+3,
   |  |  |    data_start+4 };
19 |  |  |  normal write operation;
20 |  |  else
21 |  |  end
22 |  |  normal write operation;
23 |  end
24 end
25 until end of execution  swap (instruction_way_list,data_way_list)
26 {
27 foreach  instruction_way_list[i], 0 <= i < num_inst_ways
28  do
29 |  temp=instruction_way_list[i];
30 |  instruction_way_list[i]=data_way_list[i];
31 |  data_way_list[i] =temp ;
32 |  } t_p=0;
33 end
```

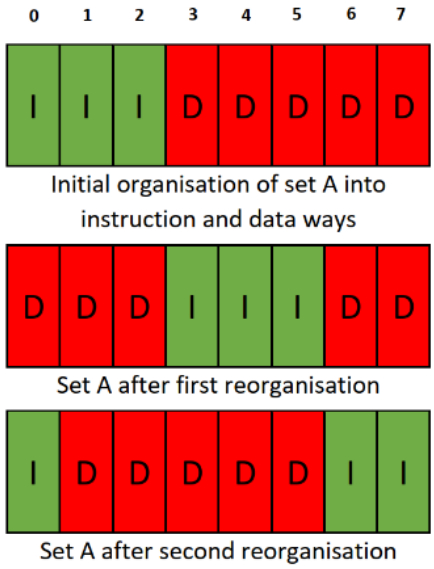

Figure 5.3: Organization of a set of 8-way set associative cache in ViSC architecture

stems from the observation that data typically requires more space in unified caches due to its larger memory footprint. In ViSC, owing to its logical partitioning, each cache set comprises three instruction ways for storing instructions fetched from main memory, leaving the remaining five ways for data storage. The mapping of instruction and data ways during the initial phase and subsequent reorganizations is illustrated in Figure 5.3. Initially, during cache warm-up, way 0, way 1, and way 2 serve as instruction ways, housing instruction blocks mapped to this set, while the remaining ways store data. We establish a threshold time of 100,000 clock cycles. Upon its expiration, reorganization occurs, and ways 3, 4, and 5 become the instruction ways, as depicted in the figure. Similarly, instruction and data ways periodically shift across the 8-ways at regular intervals.

All read requests to the LLC are handled as usual. However, regarding write requests, ViSC assesses whether the elapsed time since the last cache reorganization surpasses a predetermined threshold. If this condition is met, a set reorganization process is triggered, wherein the contents of the current instruction/data way are swapped with their counterparts in the new data/instruction way. This swapping and copying procedure within the L2 cache occurs in the background and does not interfere with the critical path of instruction execution. The processor continues its execution

by retrieving instructions and data from the L1 cache. In ViSC, instruction ways are written only while transferring a block from the main memory to the L2 cache. On the other hand, data ways are written during L2 cache misses and subsequent block transfers from main memory and during write-back or write-through operations from the L1 cache. Evictions of L1-I cache blocks do not trigger the write operations in the L2 cache. However, the removal of dirty cache blocks from the L1 cache significantly contributes to writes on the data ways of the L2 cache in ViSC. By reorganizing instruction (I) and data (D) ways in ViSC, the distribution of writes across each way in a cache set is balanced. As previously mentioned, in a cache with m-way set associativity, there are various possible logical configurations for the split between instruction ways *(k)* and data ways *(m-k)*.

Figure 5.4 shows relative lifetime values of different partition ratios of ViSC enabled L2 cache for various benchmarks. It is observed that for an 8-way set associative cache the setting k=3 yields optimal performance. ViSC presents a promising proactive strategy for managing cache aging. Unlike reactive approaches that only activate wear-leveling under specific conditions, ViSC's proactive method ensures continuous wear-leveling regardless of application behaviour. This eliminates the need for write counters, minimizing overhead. However, ViSC employs static values for crucial parameters like reorganization interval and instruction to data partition ratio. Figure 5.5 shows the analysis of relative lifetime values of different reorganization of ViSC enabled L2 cache for various benchmarks. We can observe that the benchmarks with high write accesses necessitate frequent reorganization for optimal write distribution, while others exhibit sudden bursts of writes, overwhelming specific memory cells. ViSC's limited number of ways allocated to data in LLC sets may penalize data-intensive applications by constraining effective storage capacity. These challenges drive the need for a dynamic, application-aware policy to adjust critical parameters based on application behaviour. To address this, we propose two low-overhead techniques, E-ViSC and P-ViSC, which augment conventional ViSC capabilities. These techniques employ a hybrid wear-leveling policy triggered regularly (proactive) and dynamically adjust wear-leveling frequency by analyzing application write patterns (reactive). Detailed discussions on these proposed techniques follow in

subsequent sections.

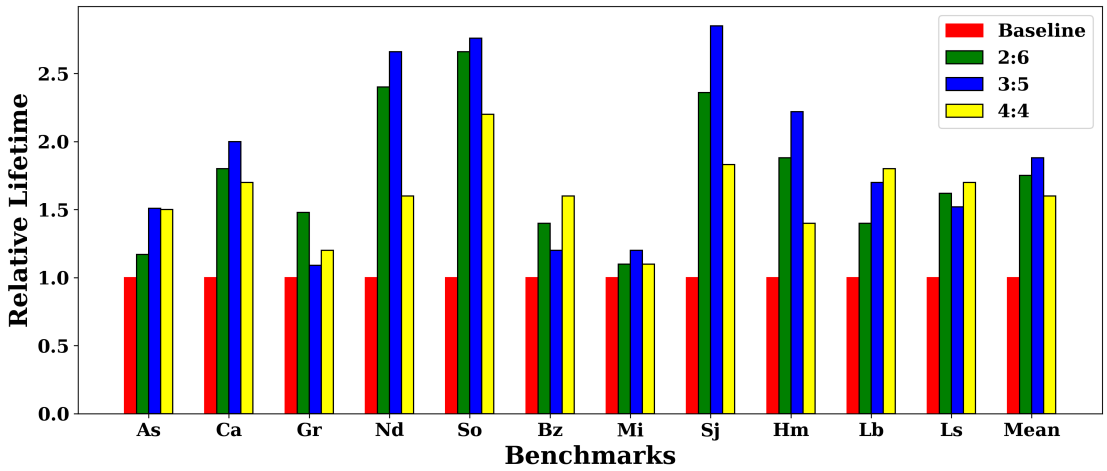

Figure 5.4: Relative lifetime values of different partition ratios of ViSC enabled L2 cache for various benchmarks

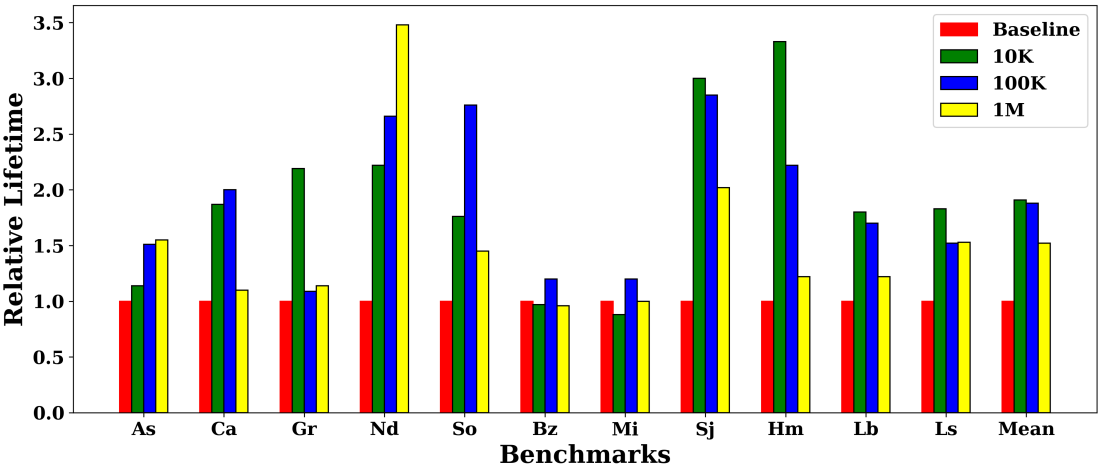

Figure 5.5: Relative lifetime values of different reorganization intervals (cycles) of ViSC enabled L2 cache for various benchmarks

5.3.2 Enhanced-ViSC

Enhanced-ViSC (E-ViSC) operates by dividing a unified m-way LLC into distinct sections, reserving n-ways for instructions and allocating the remaining $(m\text{-}n)$ ways

for data storage. This allocation is adjusted periodically, ensuring that instructions and data are balanced optimally. As instruction blocks are less frequently modified than data blocks, this periodic adjustment aids in more evenly distributing write operations. E-ViSC dynamically adapts the frequency of these adjustments based on the LLC's write behaviour, utilizing a global counter to track total writes across the LLC within specific time intervals. Unlike conventional methods that track writes at the block level, E-ViSC analyzes writes at the LLC level, sacrificing some precision for overhead savings.

Detailed workings of E-ViSC are described in Algorithms 2 and 3. Algorithm 2 introduces the key parameters and functions used in Algorithms 3 and 4. Algorithm 3 introduces dynamic reorganization based on elapsed time (t_p) and write thresholds (T_i). Algorithm 3 initializes the reorganization interval (R_i) and threshold (T_i) to their smallest values in line numbers 1–2. For every cache request, read requests (line number 4) perform normal operations, while write requests trigger reorganization checks. If the time elapsed exceeds the reorganization interval (t_p $>$ R_i) in line number 7, the swap function in line number 8 is executed, followed by set_reorganize in line number 9 and a normal write operation follows. The algorithm dynamically adjusts R_i based on the current write_count and thresholds (T_i) in line numbers 10–17, ensuring that the cache adapts to varying workloads. The loop repeats until execution ends (line number 20), maintaining a balance between instruction and data ways while minimizing wear.

Let us examine the working of E-ViSC in detail. Initially, the reorganization interval is set to 100K cycles, with a fixed instruction-to-data partition ratio of 3:5. Each LLC write operation increments a 10-bit global counter, triggering a set reorganization once the interval lapses. Reorganization occurs only upon the first write to a set following the interval's completion, ensuring uninterrupted cache operation. A 64B write buffer facilitates partition adjustments by swapping contents. Based on the normalized write count within the current interval, E-ViSC selects a new reorganization interval from a predefined pool of standard intervals (25K, 50K, 75K, 100K, and 1M cycles). Opting discrete values for reorganization interval simplifies complexity and reduces overhead. Figure 5.6 illustrates the configuration of set A

in an E-ViSC-enabled 8-way set associative cache across different execution phases. E-ViSC prolongs reorganization intervals during low-write phases to delay swaps and associated writes. However, E-ViSC dynamically shortens intervals as write frequency increases, leading to more frequent reorganizations and improved write distribution. This dynamic adaptation of reorganization intervals yields superior performance to traditional ViSC, as discussed in section 5.4.

ALGORITHM 2: Common terms and swap function for E-ViSC and P-ViSC

```
1  cache_assoc = A : Cache associativity;
2  num_inst_ways =N_i : Number of instruction ways;
3  num_data_ways =A − N_i : Number of data ways;
4  instr_way_start = 0 : First way among instruction ways;
5  data_way_start = N_i : First way data data ways;
6  write_count=0 : Number of writes;
7  t_p: Time elapsed since last set reorganization, increments every clock cycle;
8  R_i ∈ {R_1, ...R_n} : Set of reorganization intervals R_1.. < R_i... < R_n;
9  T_i ∈ {T_1, ...T_n} : Set of write thresholds T_1.. < T_i... < T_n;
10 List instr_way_list : List of ways reserved for instructions sequentially from
     instr_way_start. Size = 3k, k ∈ ℕ;
11 List data_way_list : List of ways reserved for data; sequentially from
     data_way_start. Size = 5k, k ∈ ℕ;
12 data_count=0 : number of data writes;
13 threshold = 100000;
14 T : Maximum value of data_count;
15 p_0, p_1 = 0 : Number of consecutive reorganisation intervals;
16 s : Maximum number of consecutive reorganisation intervals;
17 swap(swap) { foreach instruction_way_list[i], 0 <= i < num_inst_ways do
18 |   temp=instr_way_list[i];
19 |   instr_way_list[i]=data_way_list[i];
20 |   data_way_list[i] =temp;
21 t_p=0;
22 }
23 set_reorganize(set_reorganize)
24 {
25 instr_way_start = (instr_way_start+num_inst_ways) % cache_assoc;
26 data_way_start = (data_way_start+num_inst_ways) % cache_assoc;
27 instr_way_list = {instr_way_start,...,instr_start + num_inst_ways − 1};
28 data_way_list = {data_way_start,... data_num_data_ways − 1};
29 }
```

ALGORITHM 3: Operational steps in E-ViSC module

```
1  R_i = R_1;
2  T_i = T_1;
3  repeatfor every LLC request R and block B do
4      if R==read then
5          normal read operation;
6      else
7          if t_p > R_i then
8              swap();
9              set_reorganize(); normal write operation;
10             if write_count < T_i then
11                 R_i = R_1
12             if write_count >= T_i AND write_count < T_{i+1} then
13                 R_i = R_{i+1}
14             if write_count >= T_{i+1} AND write_count < T_{i+2} then
15                 R_i = R_{i+3}
16             ...if write_count >= T_n then
17                 R_i = R_n
18         else
19             normal write operation;
20 until end of execution
```

5.3.3 Protean-ViSC

P-ViSC, like E-ViSC, operates by virtually partitioning the unified cache into separate data and instruction caches. This involves dividing an m-way LLC into n-ways for instructions and *(m-n)*-ways for data. Notably, the number of writes in instruction blocks is considerably lower than in data blocks. While a fixed partition ratio may penalize applications, especially those with either data or instruction write-intensive operations, P-ViSC addresses this by dynamically managing the partition ratio based on the application's write count. To achieve this, P-ViSC utilizes a 10-bit saturating global counter, *data_count*, which increments upon data writes and decrements upon instruction writes in the LLC. This counter determines whether write accesses to the LLC predominantly involve instruction or data blocks. Initially, the instruction-to-data way ratio is set at 3:5, with a reorganization interval of 100K cycles. Following each interval, if *data_count* stabilizes between 0 and 1023, indicating balanced write patterns, no changes to the partition ratio occur. However, if *data_count* deviates

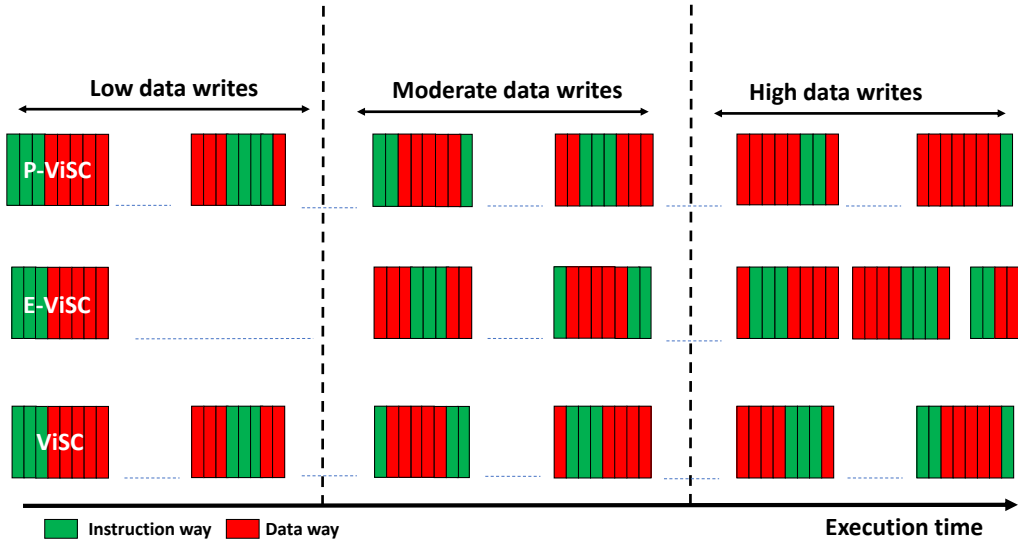

Figure 5.6: Organisation of set A of eight-way set associative caches after each reorganization of proposed techniques

from this range, signalling a need for adjustment, P-ViSC initiates a reorganization process. For instance, if *data_count* reaches 0 or 1023 consistently over p consecutive reorganization intervals, indicating a discrepancy in write patterns, P-ViSC adjusts the partition ratio accordingly. It increases instruction ways by converting data ways into instruction ways when *data_count* is 0 for p intervals, or vice versa when *data_count* saturates to 1023. This realignment allows for accommodating more blocks of the prevalent type. After each reorganization, the cache operates with the new partition ratio and repeats the process periodically. The criterion of p consecutive intervals ensures stability around the 3:5 partition ratio unless continuous application behaviour necessitates otherwise. Additionally, P-ViSC ensures that at least one way is reserved for instructions. The decision-making process for partition ratio reorganization, guided by the global counter *data_count*, reduces the overhead of estimating writes at the way-level granularity. Depending on the write pattern to NVM, P-ViSC dynamically adjusts its instruction-to-data partition ratio. This adjustment enables the accommodation of more data writes by increasing the number of data blocks.

Algorithm 4 explains the working of P-ViSC. Cache requests are handled similar to E-ViSC, with read operations following normal procedures (line number 4) and write

operations triggering reorganization if t_p > threshold (line number 6). In this case, the swap function and set_reorganize are executed in lines 7–8, followed by a normal write operation (line number 9). Depending on data_count, adjustments are made to the number of instruction ways. If data_count ≤ 0, counters p_0 is incremented and and p_1 is reset (line numbers 10–12), potentially increasing instruction ways. Conversely, when data_count $\geq$T, p_1 is incremented, and instruction ways may decrease if conditions are met (line numbers 22–27). Default conditions reset the counters in lines 29–30, and data_count is updated based on the type of write operation (lines 33–36). Algorithm 5 shows the increment and decrement operations on instruction way list in P-ViSC.

Table 5.1: Comparison of the proposed wear-leveling techniques

	Reorganization interval	Partition ratio	Application awareness
ViSC	Static	Static	No
E-ViSC	Dynamic	Static	Yes
P-ViSC	Static	Dynamic	Yes

Table 5.1 summarises the key features of ViSC, E-ViSC and P-ViSC such as reorganization interval, partition ratio and application awareness. In ViSC reorganization and partition ratio are static where as in E-ViSC, the reorganization interval and in P-ViSC, partition ratio are dynamic in nature. The third feature, application awareness refers to the ability of the proposed technique to adjust it self to suit according to the application behaviour. Further enhancements, such as combining the dynamic reorganization interval and partition ratio adjustment seen in both E-ViSC and P-ViSC, are possible for ViSC. However, experimental results suggest that while such a combination yields performance similar to E-ViSC, the overhead limits significant improvements. Therefore, the discussion primarily focuses on the effectiveness of E-ViSC and P-ViSC individually.

ALGORITHM 4: Operational steps in P-ViSC module

```
1  repeat
2  for every LLC request R and block B do
3      if R==read then
4          normal read operation;
5      else
6          if t_p > threshold then
7              swap();
8              set_reorganize();
9              normal write operation;
10             if data_count <= 0 then
11                 p_0 + +;
12                 p_1 = 0;
13                 if num_inst_ways <= num_data_ways then
14                     if num_inst_ways%3 == 0 then
15                         if p_0 == s then
16                             inst_way_increment();
17                         else
18                             inst_way_increment();
19             else if data_count >= T then
20                 p_1 + +;
21                 p_0 = 0;
22                 if num_inst_ways>=1 then
23                     if num_inst_ways == 3 then
24                         if p_1 == s then
25                             inst_way_decrement();
26                         else
27                             inst_way_decrement();
28             else
29                 p_0 = 0
30                 p_1 = 0;
31         else
32             normal write operation;
33             if data_write then
34                 data_count + +;
35             else
36                 data_count − −;
37 until end of execution
```

ALGORITHM 5: P-ViSC :inst_way_increment() and inst_way_decrement()

```
1  inst_way_increment(inst_way_increment)()
2  {
3  num_data_ways − −;
4  num_inst_ways + +;
5  data_count = 0; }
6  inst_way_decrement(inst_way_decrement)()
7  {
8  num_data_ways + +;
9  num_inst_ways − −;
10 data_count = 0; }
```

5.4 Experimental Setup and Result Analysis

Our evaluation of proposed architectures involves running workloads composed of SPEC CPU2006 benchmark applications using the system configurations outlined in Table 5.2. We compare the proposed techniques with baseline architecture and DWWR. Baseline architecture refers to the SLC NVM L2 cache without any optimization. DWWR is an optimization technique for SLC NVM caches which uses a dynamic window write restriction policy. A detailed description of DWWR can be found in Section 2.2.

Table 5.2: Simulation parameters

CPU	2 GHz, Unicore, Dual-Core, Quad- Core, ALPHA
L1 Cache	Private, 32KB, SRAM Split cache, 64 B block, 4-way set associative
L2 Cache	Shared 512KB|2MB, STT-RAM cache, 4-way|8-way|16-way set associative, 64 B block
Main Memory	8 GB

5.4.1 Impact on Relative Lifetime

Relative lifetime is one of the most widely used metrics that can measure the aging of NVMs [12] [11] [13]. As discussed before, limited endurance is a bottleneck for NVMs. Attackers can also exploit this limitation to run applications that force more writes on certain blocks of NVM LLC. In order to reduce the concentration of writes to a few memory locations, we need to distribute the writes as much as possible to

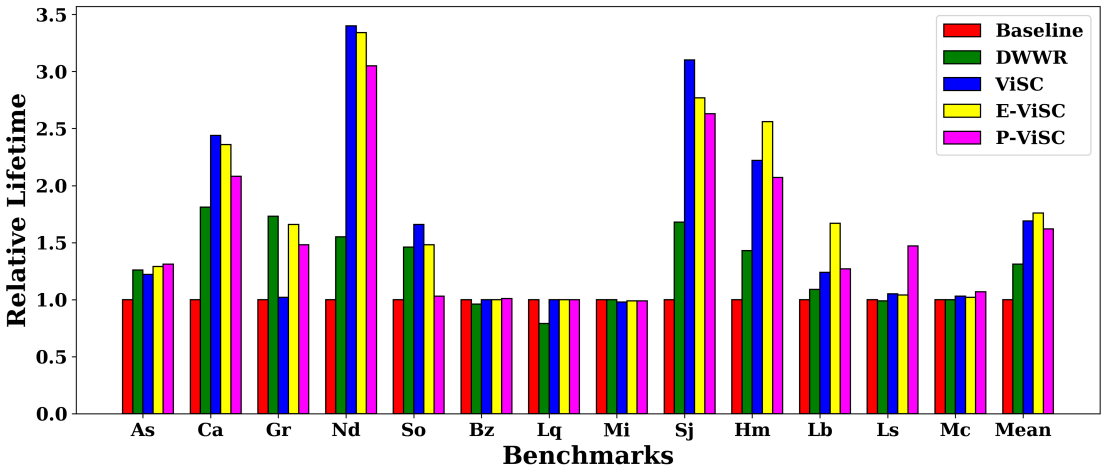

Figure 5.7: Relative lifetime of 512KB 8-way set associative NVM L2 cache in a unicore system

avoid the early aging of memory cells. If an application produces concentrated writes to a particular block, then it reduces the relative lifetime. We measure the relative lifetime of NVM L2 cache in a unicore, dual-core and quad-core system using various applications from the SPEC CPU2006 benchmark suite.

In the case of the unicore system illustrated in Figure 5.7, it is evident that all three variants; ViSC, E-ViSC, and P-ViSC substantially enhance the lifetime compared to the conventional ViSC. Both E-ViSC and P-ViSC exhibit respective improvements in relative lifetime by 1.71x and 1.57x, while ViSC itself boosts the relative lifetime by 1.64x compared to the baseline across a range of benchmark categories. The adaptability to self-adjust reorganization intervals notably enhances the lifetime in E-ViSC, especially evident in benchmarks with high WPKI values such as *hmmer* and *lbm*, where frequent reorganizations are necessary for optimal wear-leveling; a feature supported by E-ViSC. Conversely, for benchmarks with low WPKI, ViSC outperforms E-ViSC. An examination of E-ViSC's execution time distribution (Table 5.4) reveals that applications with low WPKI tend to settle on a reorganization interval of 1M, starting from an initial 100K interval. Since the write count within this interval falls below the threshold, the system transitions to a larger interval, leading to increased write variance across cache blocks, affecting relative lifetime. ViSC's operation at a fixed, shorter reorganization interval ensures

a more balanced write distribution across cache blocks. Similarly, the dynamic reorganization of instruction and data ways' position and count based on application behaviour enhances P-ViSC's performance. Notably, benchmark *leslie3d* demonstrates significant improvement in relative lifetime due to P-ViSC's adaptive nature. Analysis reveals that the instruction-to-data mapping shifts from 3:5 to 2:6 and then to 1:7 to accommodate *leslie3d*'s larger data memory footprint, increasing data blocks in L2 cache and reducing frequent writes from heavy L2 cache misses. Detailed examination of P-ViSC's execution time distribution (Table 5.5) indicates a preference for 3:5 and 4:4 partition ratios in low WPKI benchmarks, indicating lesser data storage requirements in L2 cache. However, this can lead to uneven writes across ways, occasionally reducing P-ViSC's relative lifetime compared to ViSC, particularly in scenarios where way reorganization affects write balance negatively for specific low WPKI benchmarks.

We also investigate the effectiveness of the proposed methodologies in dual-core systems that share NVM L2 cache. In the context of dual cores, we execute two independent applications (one per core) and evaluate their combined impact on the L2 cache. The influence on the L2 cache from write patterns generated by one core may be enhanced or counterbalanced by the write patterns from the other core. To comprehensively understand this impact across various application-level combinations, we devise six workloads, each comprising a blend of two independent SPEC CPU2006 benchmark applications. The initial three workloads (designated as Low, Mid, and High) consists of two independent applications with similar WPKI characteristics. Additionally, we formulate three mixed-category workloads (Low-Mid, Low-High, and Mid-High) by pairing applications from different categories.

In Figure 5.8, we present a comparative analysis of the relative lifetime in dual-core systems. ViSC, E-ViSC, and P-ViSC enhance the lifetime by 1.81x, 2.03x, and 1.57x, respectively, across diverse benchmark mix categories compared to the baseline system. Notably, except for the Low category, E-ViSC demonstrates superior performance to ViSC. Upon examining the distribution of execution time of E-ViSC in dual-core systems (refer to Table 5.4), we observe that the reorganization interval predominantly falls within 50K or 25K for most workloads. This adaptability stems

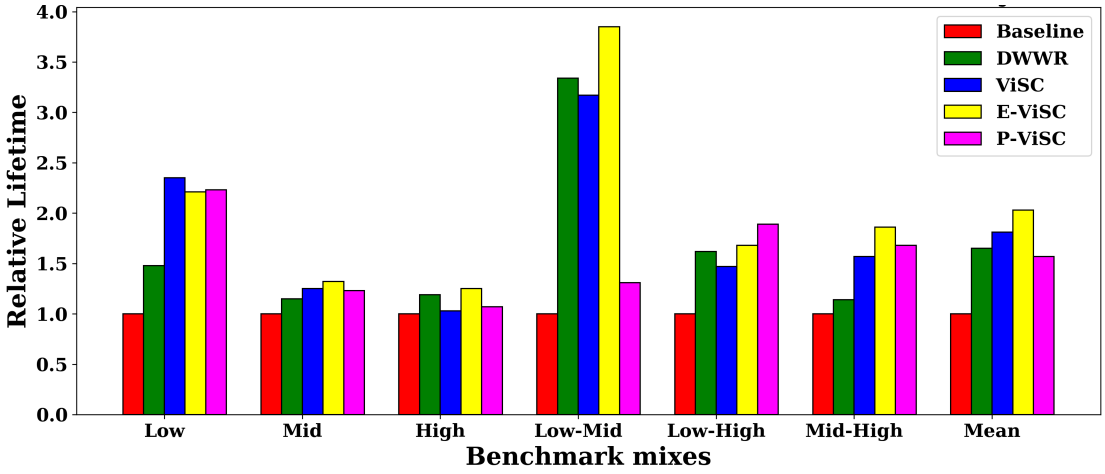

Figure 5.8: Relative lifetime of 512KB 8-way set associative NVM L2 cache in a dual-core system

from the adjustment of the reorganization interval based on L2 cache writes. Given the identical L2 cache size of 512KB in dual-core setups, there are more writes and evictions in the L2 cache compared to single-core systems. Consequently, the E-ViSC mechanism adapts to a more frequent reorganization interval, leading to improved relative lifetime by reducing uneven writes across cache blocks.

We conducted studies on quad-core systems, similar to those performed on dual-core systems, utilizing various workloads. In selecting benchmark mixes, we utilized two distinct benchmarks from the first and second WPKI categories.

Figure 5.9 illustrates the analysis of the relative lifetime of quad-core systems. Like with dual-core systems, ViSC demonstrates superior performance with low WPKI category benchmarks. Our strategies prove effective, particularly when encountering significant write variations. In quad-core systems, intra-set variation appears reduced as the diverse write patterns from four different cores ultimately balance out concentrated writes to specific blocks. P-ViSC exhibits the best performance for high WPKI category benchmarks among the four techniques examined. Within the execution window, benchmarks in the high WPKI category exhibit a notably higher miss rate than others. Due to its ability to accommodate the maximum number of data blocks throughout the execution window, P-ViSC effectively reduces writes resulting from L2 replacements.

We study the impact on relative lifetime of proposed techniques against other

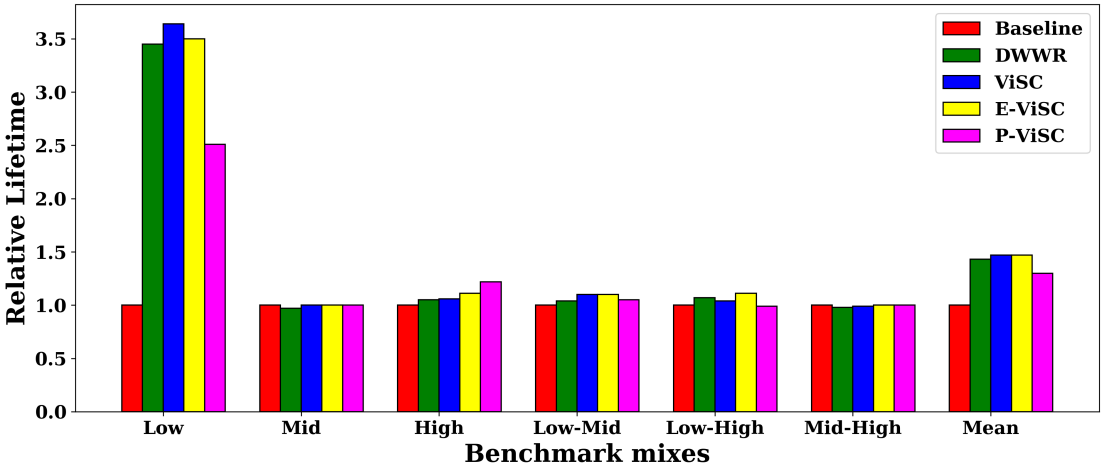

Figure 5.9: Relative lifetime of 512KB 8-way set associative NVM L2 cache in a quad-core system

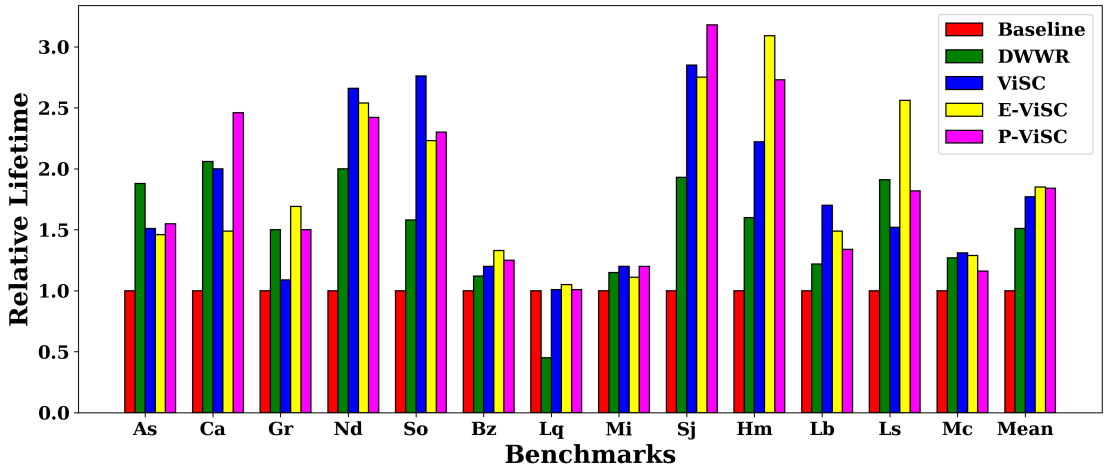

Figure 5.10: Relative lifetime of 2MB 8-way set associative NVM L2 cache in a unicore system

wear-leveling methods using a larger L2 cache of 2MB. This ensures the scalability of our technique for various cache sizes. Similar to that of a 512KB L2 cache, impact of balancing and write distribution becomes more pronounced with the higher cache capacity, attributed to the increased number of cache blocks. As cache size increases cache replacement due to capacity misses reduces, thereby enhancing block retention and increasing cache intra-set variation due to a greater number of dead blocks. In

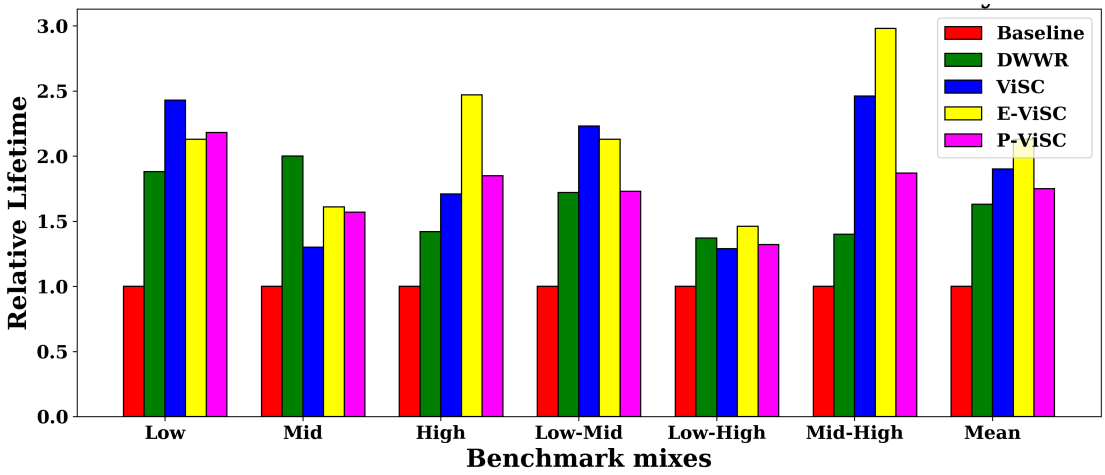

Figure 5.11: Relative lifetime comparison of 2MB 8-way set associative NVM L2 cache in a dual-core system

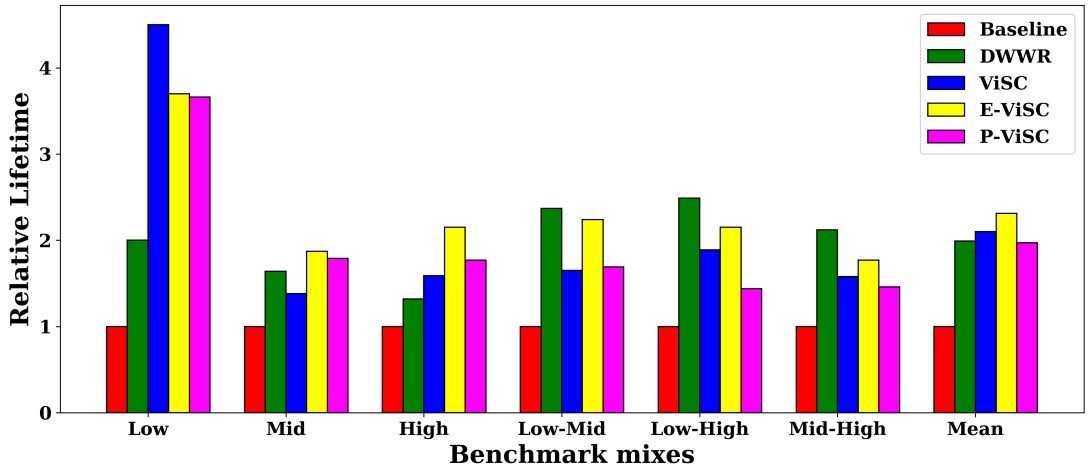

Figure 5.12: Relative life time comparison of 2MB 8-way set associative NVM L2 cache in a quad-core system

a unicore system utilizing a 2MB NVM L2 cache, E-ViSC and P-ViSC elevate the relative lifetime by 1.85x and 1.84x, respectively, compared to the baseline, with ViSC achieving a 1.77x improvement. The outcomes align with those observed with a 512KB cache size. In a dual-core system equipped with a 2MB L2 cache, E-ViSC demonstrates a 23% enhancement in relative lifetime compared to ViSC, while P-ViSC exhibits notable performance improvements across various benchmark mixes. Mainly, P-ViSC shows approximately a 5% better relative lifetime for the Low benchmark mix compared to E-ViSC. In quad-core systems featuring a 2MB L2 cache, both

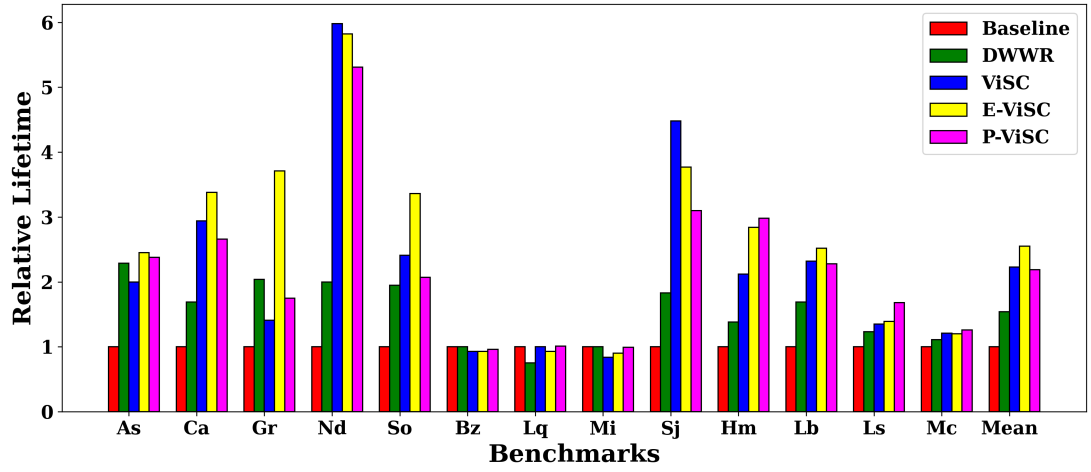

Figure 5.13: Relative lifetime of 512KB 16-way set associative NVM L2 cache in a unicore system

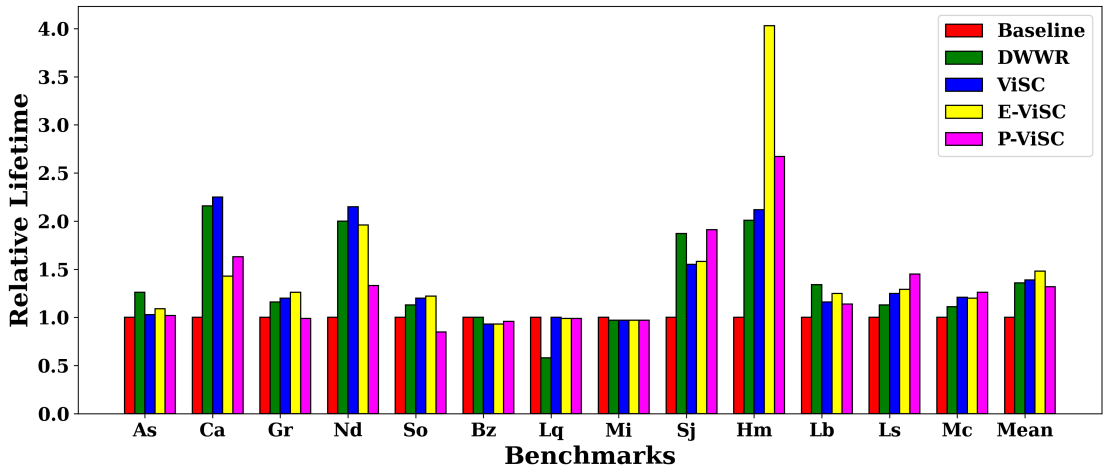

Figure 5.14: Relative lifetime of 512KB 4-way set associative NVM L2 cache in a unicore system

E-ViSC and P-ViSC demonstrate superior performance across all combinations except for the Low category benchmark. The proposed techniques exhibit relative lifetime improvements in larger caches for a given system architecture due to the potential for enhanced write redistribution compared to smaller caches.

We study the impact of associativity of a given cache size on the relative lifetime improvement of our proposed technique. Figures 5.13 and 5.14 illustrate the relative lifetime of 16-way and 4-way 512KB L2 caches across various architectures. For the 16-way E-ViSC enabled L2 cache, employing six instruction ways and ten data ways,

ViSC, E-ViSC, and P-ViSC increase the relative lifetime by 2.23x, 2.55x, and 2.29x, respectively. Conversely, for the 4-way L2 cache with two instruction and two data ways, the relative lifetime increases by 1.34x, 1.48x, and 1.32x, respectively. The relative lifetime improvement escalates as set associativity increases for a particular cache size, facilitating better write masking and inclusion of data blocks due to the expanded number of ways.

5.4.2 Impact on Intra-set variation

Intra-set variation quantifies the efficiency of the wear-leveling method in distributing write operations within the cache memory sets. The coefficient representing intra-set variation, denoted as $IntraV$ (as defined in equation 3.1), provides insights into the uniformity of write distribution across different cache set ways. Lower values of $IntraV$ indicate a more balanced write distribution.

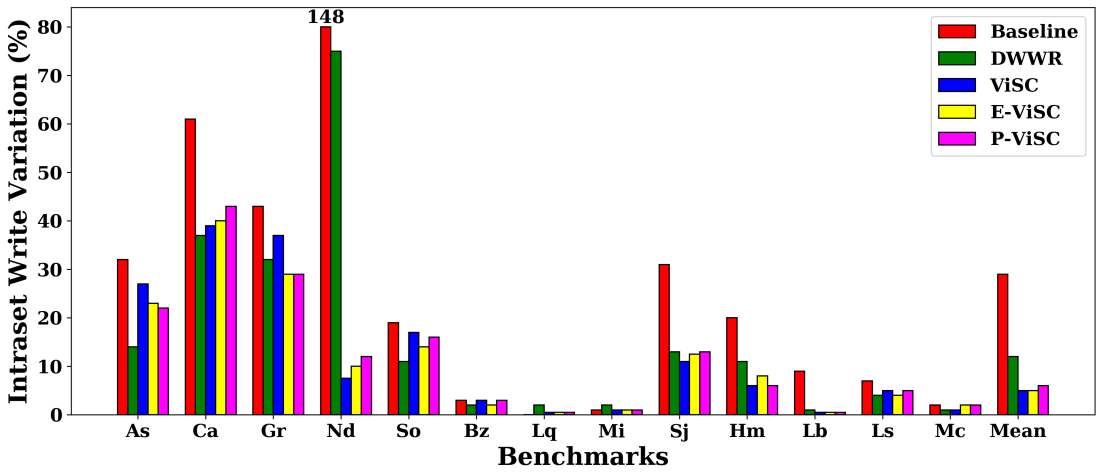

Figure 5.15: Intra-set variation comparison of 512KB 8-way set associative NVM L2 cache in a unicore system

Figure 5.15 illustrates the write distribution for a uni core system utilizing a 512KB L2 cache. Evidently, applications with low WPKI values, such as *namd* and *calculix*, exhibit significant intra-set variation compared to those with mid and high WPKI values. These low WPKI applications demonstrate uneven write patterns across the cache ways, with certain ways heavily utilized over others. Conversely,

applications like *libquantum* and *mcf*, characterized by high WPKI values, display a more uniform distribution of writes across cache ways. Despite notable reductions in intra-set variation compared to the baseline, both E-ViSC and P-ViSC outperform other techniques, indicating their ability to balance effectively writes across cache ways. Specifically, E-ViSC achieves a 4% greater intra-set variation reduction than ViSC.

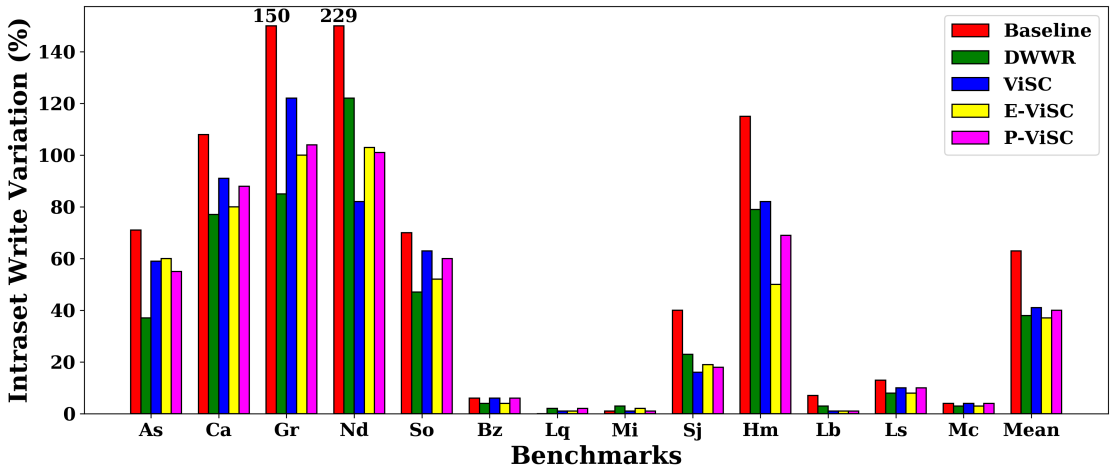

Figure 5.16: Intra-set variation comparison of 2MB 8-way set associative NVM L2 cache in a unicore system

Figure 5.16 presents the study of intra-set write variation in a unicore system employing a 2MB NVM L2 cache. As the cache size increases, so does the intra-set variation due to heightened cache block retention. Notably, the effectiveness of our techniques, particularly E-ViSC, becomes more pronounced with larger cache sizes. E-ViSC demonstrates superior performance across various applications, including *namd*, *gromacs*, *hmmer*, and *calculix*, owing to its capability for frequent reorganization, which facilitates better write distribution compared to static reorganization interval-based approaches like ViSC and DWWR

Figures 5.17 and 5.18 depict the comparison of intra-set variation in a dual-core system utilizing 512KB and 2MB L2 cache, respectively. As previously discussed regarding relative lifetime, dual-core systems exhibit diminished write variation and consequently lower $IntraV$ compared to unicore systems. The presence of two applications sharing an LLC mitigates the write imbalance and $IntraV$ to some

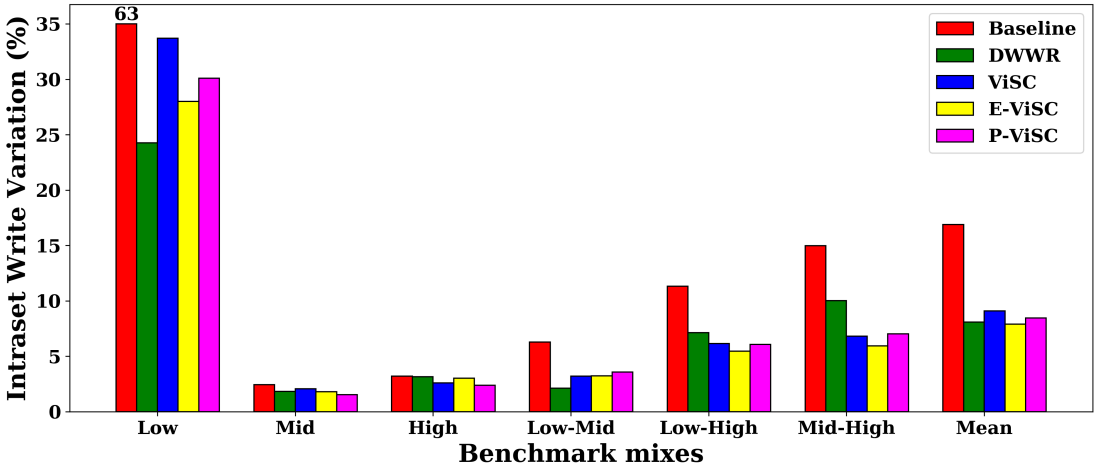

Figure 5.17: Intra-set variation comparison of 512KB 8-way set associative NVM L2 cache in a dual-core system

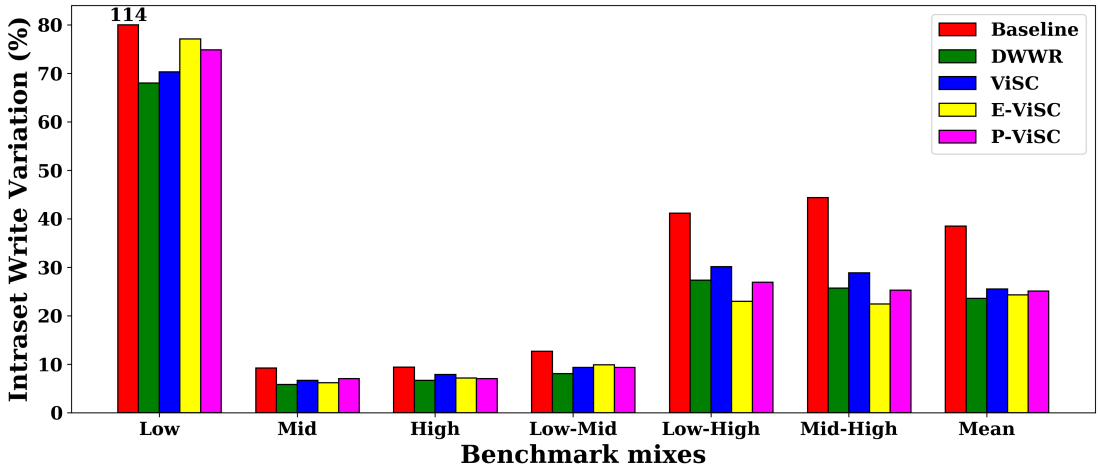

Figure 5.18: Intra-set variation comparison of 2MB 8-way set associative NVM L2 cache in a dual-core system

degree. In the case of a 512KB L2 cache, E-ViSC demonstrates a 53% reduction in intra-set write variation compared to the baseline and a 13% reduction compared to ViSC. Similarly, P-ViSC exhibits a reduction in intra-set variation by 50% and 7% compared to baseline and ViSC, respectively. Similar to unicore systems, as the cache size increases from 512KB to 2MB, the write variation also increases. This trend is evident from the results of the 2MB L2 cache in Figure 5.18.

Given that multi-core systems inherently display less write variation compared to unicore systems, the reduction in intra-set variation diminishes as the number of cores increases. Quad-core systems demonstrate superior write distribution compared to unicore and dual-core systems.

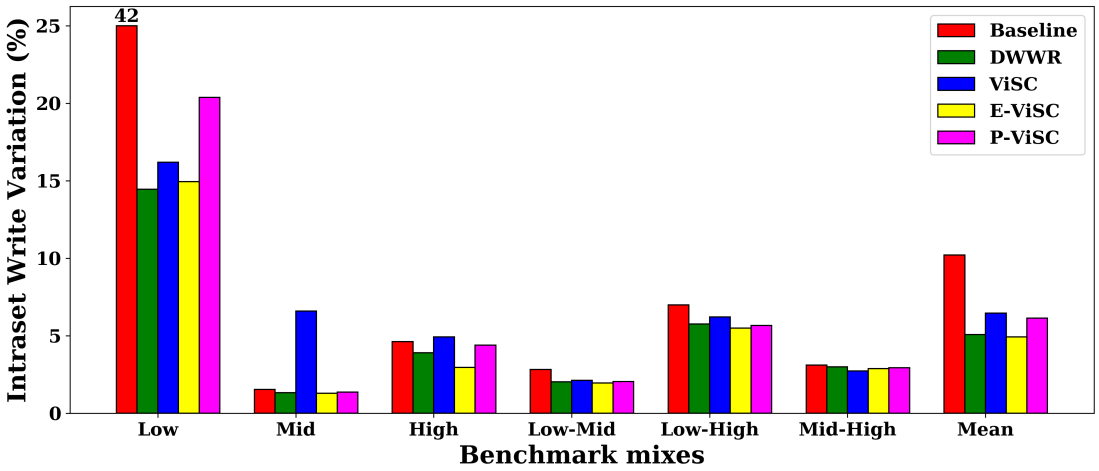

Figure 5.19: Intra-set variation comparison of 512KB 8-way set associative NVM L2 cache in a quad-core system

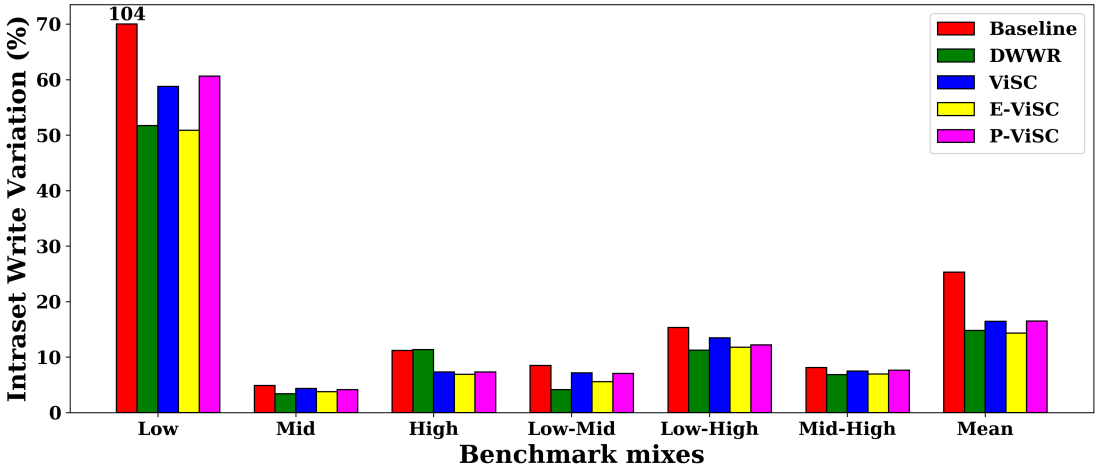

Figure 5.20: Intra-set variation comparison of 2MB 8-way set associative NVM L2 cache in a quad-core system

Figures 5.19 and 5.20 illustrate the write variation in quad-core systems utilizing 512KB and 2MB NVM caches. The results indicate that our proposed techniques

enhance intra-set variation in quad-core systems as well. As anticipated, E-ViSC exhibits the most significant reduction in write variation among all techniques. The superior write distribution of E-ViSC compared to P-ViSC suggests that application-aware dynamic reorganization interval selection yields better write distribution than dynamic partition ratio selection.

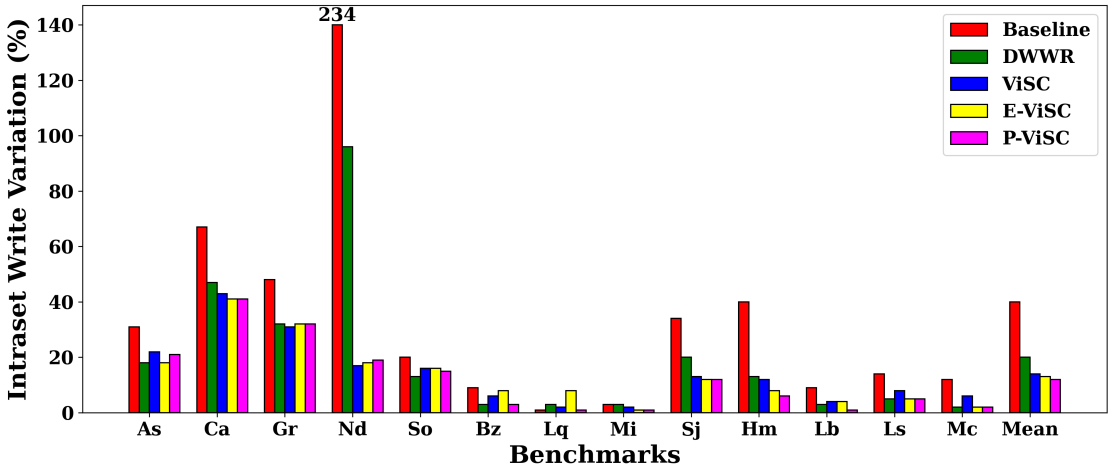

Figure 5.21: Intra-set variation comparison of 512KB 16-way set associative NVM L2 cache in a unicore system

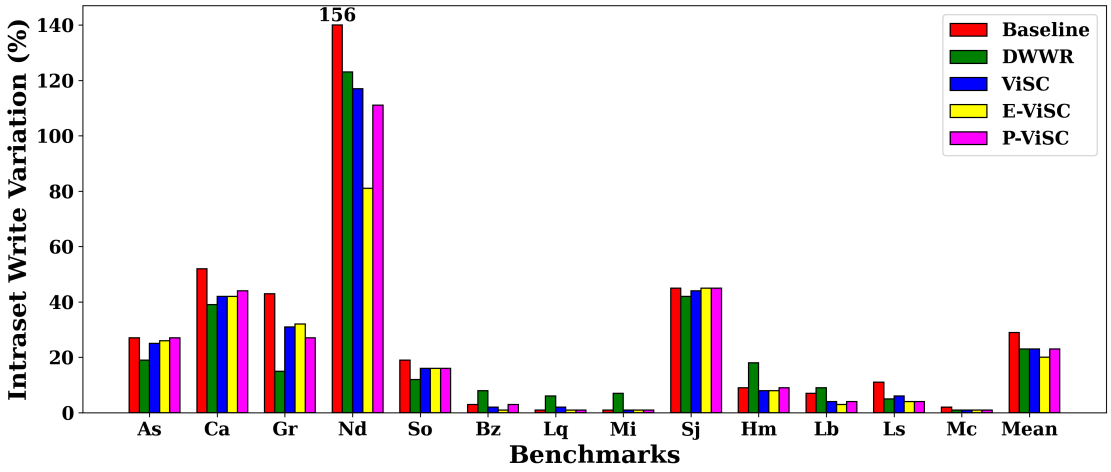

Figure 5.22: Intra-set variation comparison of 512KB 4-way set associative NVM L2 cache in a unicore system

Examining figures 5.21 and 5.22, we explore the intra-set write variation concerning a 512KB L2 cache size, comparing 16-way and 4-way set associativity. Our

Table 5.3: Percentage change in inter-set variation for various architectures and cache sizes under study

		DWWR	ViSC	E-ViSC	P-ViSC
512KB	unicore	+1.15	-1.17	-0.82	+0.90
	dual-core	-5.52	-7.78	+4.29	+0.55
	quad-core	-1.88	-3.34	-1.68	+0.94
2MB	unicore	+2.53	+1.67	+0.66	+4.46
	dual-core	+1.44	-7.20	+1.00	-5.92
	quad-core	-1.11	-1.33	-1.69	-0.96

suggested methods exhibit a more pronounced impact on intra-set variation as associativity increases. This occurs because higher associativity facilitates the more efficient distribution of writes within a set, thereby diminishing write variation.

5.4.3 Impact on Inter-set variation

Our proposed techniques aim to decrease the write variation within sets of NVM LLCs. As cache block reorganization and swapping occur within sets, there is no direct effect on the variability between sets. We examine the inter-set variability of these techniques across unicore, dual-core, and quad-core configurations using 512KB and 2MB NVM L2 caches. We observe minor fluctuations in the InterV, as detailed in Table 5.3, with changes typically in the range of a few percentage points. These fluctuations are primarily attributed to alterations in write counts within each set, contributing to the overall estimation. Consequently, we conclude that our proposed methods do not significantly influence inter-set variability, as our primary objective is to achieve a balanced distribution of writes within each set.

5.4.4 Impact on IPC

Proposed techniques do not have any impact on the cache hit rate and memory access time. Swapping operations during reorganizations are not on the critical path of instruction execution and hence do not impact memory access time. Hence the proposed techniques do not impact the instructions per cycle (IPC) value.

5.4.5 Impact on Execution Time Distribution

E-ViSC dynamically adjusts its reorganization intervals based on the write pattern behaviours exhibited by the application. Table 5.4 outlines the distribution of execution times in various architectures using a 512KB NVM L2 cache for unicore, dual-core, and quad-core systems.

In unicore setups, particularly with low WPKI benchmarks, E-ViSC often employs longer reorganization intervals, constituting approximately 53% of the total execution time on average. This delay in swapping instruction and data ways responds to fewer writes. However, this strategy may slightly reduce lifetime in scenarios with low WPKI and high write variation due to concentrated writes in fewer blocks. Instances where bursts of heavy writes occur prompt E-ViSC to adopt shorter reorganization intervals of 25K or 50K cycles, even though these are isolated cases (approximately 6% for 25K cycles and 3% for 50K cycles) to which E-ViSC adjusts accordingly.

For mid and high WPKI benchmarks, E-ViSC tends to utilize shorter reorganization intervals, enhancing write distribution compared to conventional ViSC. Around 65% and 8% of the execution window adopt 50K cycles as the mid and high WPKI benchmarks reorganization intervals, respectively. Notably, approximately 77% of the execution time employs the shortest possible interval of 25K cycles for high WPKI benchmarks. This detailed examination illustrates E-ViSC's dynamic adaptation to application behaviour.

Similar behaviour is observed in dual-core and quad-core systems. In dual-core setups, except for low WPKI benchmark mixes, lower reorganization intervals are predominantly used due to higher average write counts resulting from inter-core cache block access and memory access interference from multiple applications. In quad-core configurations, aside from low category benchmarks, E-ViSC overwhelmingly adopts the shortest reorganization interval of 25K cycles for over 98% of the execution window. Notably, there were no instances of employing a 1M cycle reorganization interval, indicating that E-ViSC's relative lifetime improvement over ViSC is from the effective dynamic selection of reorganization intervals. Various design parameters incorporated in Algorithm 3 contribute significantly to this performance enhancement.

Table 5.4: Distribution of execution time (in %) of E-ViSC while running various benchmark mixes

System	Benchmark Mix	Reorganization Interval				
		25K	50K	75K	100K	1M
Unicore	Low	5.79	3.28	9.23	28.97	52.75
	Mid	22.26	64.79	6.43	6.47	0.06
	High	77.29	7.44	2.51	9.21	3.56
Dual-core	Low	10.04	6.87	23.65	38.92	20.52
	Mid	85.71	14.25	0.00	0.04	0.00
	High	89.11	7.30	2.07	1.69	0.04
	Low-Mid	61.52	33.71	1.62	2.48	0.67
	Low-High	80.33	12.92	2.58	0.25	3.92
	Mid-High	98.62	0.42	0.41	0.47	0.08
Quad-core	Low	16.51	8.27	36.42	38.78	0
	Mid	99.98	0	0	0.01	0
	High	99.97	0.01	0	0.01	0
	Low-Mid	97.75	0.93	0.98	0.33	0
	Low-High	99.16	0.77	0.40	0.1	0
	Mid-High	99.96	0.03	0	0.01	0

We also conduct a thorough analysis of execution time variations to comprehend the changes in partition ratios when employing P-ViSC. Like ViSC, P-ViSC employs a fixed reorganization interval but dynamically adjusts the partition ratio for instruction and data paths. The distribution of average execution time for various partition ratios using P-ViSC is presented in Table 5.5.

In single-core systems, P-ViSC predominantly settles with a 3:5 partition ratio for execution across all benchmark mixes. However, for low WPKI category benchmarks, we observe that the 4:4 partition ratio also holds significant weight (38%) during execution. This is attributed to the limited number of writes in such applications, reserving five ways unnecessary for data, resulting in the conversion of one data block per set for storing instructions. As we transition to mid and high WPKI benchmarks, the proportion of execution time for 2:6 and 1:7 partition ratios increases. This correlates with the expansion of WPKI, indirectly leading to a larger memory footprint and a preference for a greater number of data blocks within a given set. The rationale behind P-ViSC settling with 3:5, 2:6, and 1:7 partition ratios for workloads

Table 5.5: Distribution of execution time of P-ViSC while running various benchmark mixes (%)

System	Benchmark Mix	Partition Ratio			
		1:7	2:6	3:5	4:4
Unicore	Low	4	2.8	55.5	37.7
	Mid	30.5	13.5	53.5	2.5
	High	26.75	13.75	58.5	1
Dual-core	Low	4.67	4.33	77.00	14.00
	Mid	25.50	15.00	57.50	2.00
	High	24.67	16.67	58.67	0.00
	Low-Mid	20.00	11.33	65.00	3.67
	Low-High	31.50	15.00	53.00	0.50
	Mid-High	28.25	17.32	54.43	0.00
Quad-core	Low	11.5	8.5	68.5	11.5
	Mid	16.5	13.5	69.5	0.50
	High	32	20.5	47.5	0
	Low-Mid	23	16.5	60.5	0
	Low-High	28	16.5	55	0.5
	Mid-High	31	17.5	51	0.5

other than entirely low WPKI benchmarks is evident in dual-core and quad-core systems. As previously discussed, a significant portion of cache block writes pertains to data block writes. Allocating a way for instructions effectively mitigates writes to that way and prolongs its lifespan. Given the fewer data writes for low WPKI applications, reserving more instruction ways helps preserve them from premature wear. Conversely, for high and mid-category benchmarks with more data writes, P-ViSC allocates additional space for data blocks by converting existing instruction ways into data ways. Therefore, we conclude that P-ViSC adapts to the dynamic write patterns of applications and adjusts the partition ratio accordingly to enhance lifetime of NVM LLCs.

5.4.6 Overhead Analysis

Most of the state-of-the art technologies typically incur substantial storage overhead due to using counters, tables, and buffers for estimating writes and managing data swapping [11] [12] [14] [13]. E-ViSC and P-ViSC employ counters, swap buffers, and

threshold timers for write counting and swapping operations, thereby contributing to the storage overhead. Let S, A, B, T, N, and M represent the quantities of sets, associativity, block size, tag size, swap buffers, and bits for counters per block, respectively. The storage overhead can be computed as follows:

$$\theta = \frac{M \times S \times A + N \times B}{S \times A(B + T)} \times 100 \tag{5.1}$$

In our experiment, we selected a buffer size (B) of 64 bytes and a 32-bit wide memory address. E-ViSC employs a swap buffer of 64 bytes, a 10-bit global saturating counter, a 10-bit threshold timer, and an additional bit per way for instruction and data way mapping, resulting in a storage overhead of 0.21%. Similarly, P-ViSC utilizes an additional 64-byte swap buffer, a 10-bit global saturating counter, a storage mechanism for partition ratio, and instruction-data mapping, resulting in a 0.22% overhead. The adoption of a global counter instead of multiple counters for each set/way significantly reduces the associated overhead. Since this overhead remains unaffected by cache size, both techniques can be applied to larger caches without incurring additional hardware overhead. Swapping operations contribute to increased write counts for the proposed techniques. In the case of the unicore system with 512KB L2 cache, ViSC exhibits a 0.40% overhead compared to the baseline architecture. For E-ViSC and P-ViSC, these overheads are 0.73% and 0.37%, respectively. E-ViSC mitigates write overhead resulting from swapping, particularly for low WPKI benchmarks, owing to longer reorganization intervals. Nevertheless, the overall write count due to swapping remains high for E-ViSC due to frequent swapping in mid and high WPKI benchmarks.

5.5 Conclusion and Future scope

This chapter introduces three techniques - ViSC, E-ViSC, and P-ViSC, that aim to enhance the lifetime of SLC NVMs when employed as LLCs. These techniques use the logical partitioning of NVM LLCs for storing data and instructions, periodically reorganizing them to distribute write operations across the memory uniformly, thereby mitigating memory cell wearout. While E-ViSC dynamically tunes reorganization

intervals, P-ViSC adjusts partition ratios during application execution. Experimental results show the efficacy of these techniques across unicore, dual-core, and quad-core systems, showcasing superior performance compared to existing methods, particularly in multi-core setups handling intensive workloads. Notably, E-ViSC exhibits better performance by adaptively capturing application-specific write patterns, while P-ViSC excels in scenarios where dynamic adjustment of instruction-data partition ratios is crucial. Furthermore, these methodologies hold promise for customization to leverage variations between hard and soft bits in Multi-Level Cell NVMs, thereby boosting LLC packaging density.

CHAPTER

6

Trace buffer Assisted Last Level Non-Volatile Caches

This chapter discusses Trace buffer Assisted Non-volatile memory Caches (TANC), a novel technique to improve the lifetime, write latency, and energy efficiency of MLC NVM LLC. Due to the structural peculiarity and limited write endurance, MLC NVMs are more susceptible to repeated write attacks and early wear out due to non-uniform writes [36] [37]. TANC uses embedded trace buffers to mitigate the effect of repeated writes on MLC NVM LLCs. TANC gives superior performance in improving the lifetime and performance of NVM LLCs when compared to other popular MLC NVM cache optimization techniques. Unlike popular lifetime improvement techniques TANC utilizes existing resources, hence the overhead is minimal.

6.1 Introduction

The previous chapters discuss lifetime enhancement techniques for SLC NVM LLCs. As discussed in previous chapters, due to the unique structure of MLC NVMs, memory cells can be of hard-bit or soft-bits. Write operations to hard-bit will result in extra

write to corresponding soft-bit owing to the soft-bit restoration. Hence the lifetime of MLC NVMs are less when compared to SLC NVMs [36]. The effectiveness of wear-leveling techniques for SLC NVMs may diminish when applied to MLC due to the inability to discern between hard and soft bits. Specialized techniques tailored for MLC caches show superior performance compared to generic approaches. Our proposed approach, known as Trace buffer Assisted Non-volatile memory Caches (TANC), employs ETB to enhance the lifetime, write latency, and energy efficiency of MLC NVM LLC. The key contributions of this chapter are as follows.

- We propose a technique, TANC and its variants, which use the Embedded Trace Buffer (ETB) to minimize the effect of repeated writes on memory cells.
- We study the impact of TANC on the lifetime, write latency, and energy of MLC NVMs with respect to SLC and other existing MLC optimization techniques.
- We study the impact of applications without cache locality and their impact on NVM LLCs and address the issues.

Before discussing TANC, we will discuss the motivation and details of ETB.

6.2 Motivation

To study the impact of restoration of soft-ways during hard-way writes on MLC NVM LLCs, we run SPEC CPU2006 benchmarks on SLC and MLC NVMs in a unicore architecture. We model a unicore system in gem5 simulator with two levels of cache and main memory. The L1-I and L1-D caches are configured as 32KB, 2-way set associative. The unified L2 cache is 512KB, 8-way set associative, and we use 8 GB of main memory. The block size is 64 bytes. We executed a 1 billion instruction window for selected benchmarks from the SPEC CPU2006 suite for SLC and MLC L2 cache and calculated the relative lifetime. Figure 6.1 shows the relative lifetime of MLC NVMs with respect to SLC NVMs for various SPEC CPU2006 benchmark programs. We can observe that MLC NVM LLC shows a lower lifetime than SLC for all benchmarks. From the experimental analysis, it is evident that for a given

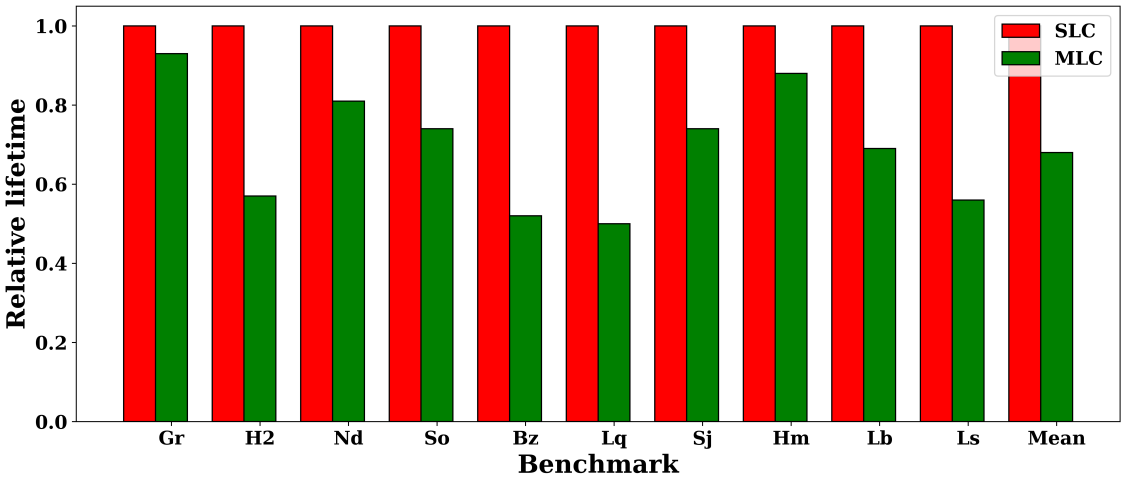

Figure 6.1: Relative lifetime of SLC and MLC NVM LLC for various SPEC CPU2006 benchmarks

application execution window, MLC NVM incurs more writes than SLC NVM caches due to soft-way restoration.

Hence, MLC NVMs are prone to early wear-out due to applications with non-uniform write applications and targeted repeated write attacks through malicious applications. The lifetime improvement techniques for SLC NVMs are unsuitable for MLC NVMs as they do not distinguish between hard and soft ways. This motivated us to develop an efficient, low-overhead technique that would enable us to use MLC NVM as LLCs, addressing the major challenges such as lower lifetime, higher write energy and latency. We discuss our proposed technique TANC and its variants in the coming sections in detail.

6.3 Proposed Techniques

We propose an innovative technique, TANC, which uses ETBs to improve the lifetime and performance of MLC NVM LLCs. Unlike the techniques discussed in the previous chapters, TANC is customized for MLC NVMs considering its unique structure. Before discussing the details of the technique, we discuss ETB, which is the core component of TANC and its variants.

6.3.1 Embedded Trace Buffer (ETB)

Embedded Trace Buffers (ETBs) [38] are available in modern processors for post-silicon validation but often remain unused after this stage. Post-silicon validation involves testing a chip for complete functional correctness in a laboratory setting, where the accuracy of the design is verified on real hardware in a practical environment. The primary goal of post-silicon validation is to identify design bugs overlooked during pre-silicon validation and to ensure the chip or System-on-Chip (SoC) is ready for deployment. An Embedded Trace Macrocell (ETM) is a real-time trace module that traces the instructions and data of a processor. Retrieving trace data from a chip using ETM becomes more complex as processor speeds increase, causing challenges in maintaining signal quality or accommodating numerous trace port pins. To address this, an on-chip buffer region is integrated to store trace data, which is later accessed at a slower rate. The data generated by the ETM is stored in the ETB, with its size varying depending on the processor architecture, ranging from 8 to 32KB. Processors like ARM 7-9 [39], Cortex A8-A9 [40], CortexM0-M3 [41], and SPARC Leon3 [42] utilize ETBs for post-silicon validation. However, after this stage, ETBs often remain unused. Given their idle status, incorporating ETBs into the memory hierarchy is proposed to reduce LLC writes, thereby enhancing performance. Various optimization techniques utilize ETBs, such as opportunistic caching of evicted blocks in Network-on-Chip (NoC) routers, which aims to reduce L1 cache miss penalties [43]. While using ETBs as victim caches for L1 D-cache can improve performance [44], it may increase area and power consumption due to additional circuitry. Our proposed technique, TANC, employs ETBs as write buffers for enhancing the performance of MLC NVM LLCs. TANC is more area and power-efficient than hybrid caches, as it utilizes existing resources and fewer SRAM components, storing hot data blocks in the SRAM portion.

6.3.2 TANC Organization

Key elements of TANC include the ETB [38], wear-leveling module, and skip module. The diagram in Figure 6.2 illustrates the memory hierarchy with TANC, while Figure 6.3 showcases the schematic of TANC-enabled L2 cache. ETB is closely linked with

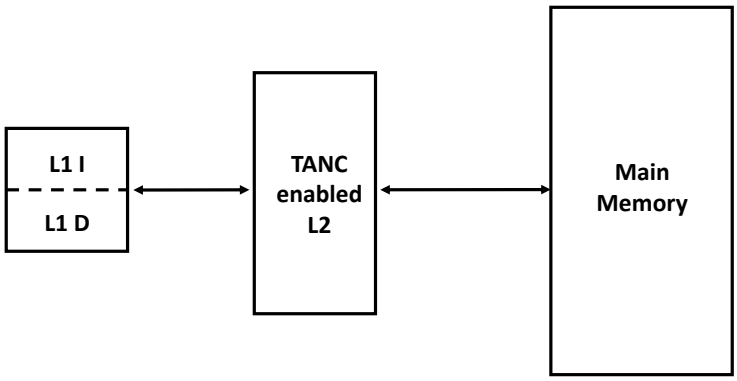

Figure 6.2: Schematic diagram of memory hierarchy with TANC

the L1 cache, and adjustments are made to the L2 cache controller to facilitate ETB access from both the L2 cache and main memory. Following post-silicon validation, control signals from the L2 cache controller enable the ETB to serve as an additional storage buffer for L2 writes. This ensures that the ETB can effectively be utilized for regular operations and validation. A segment of available ETB is utilized to buffer writes in TANC. Our experimental study show that increasing the active ETB size reduces effective writes in the MLC NVM cache enhancing both its lifetime and performance. However, incorporating more SRAM based ETB incurs higher static power dissipation. As shown in Table 2.1 SRAM cells consume 60nW per cell as leakage power. This is because the SRAM cell usually operates in standby mode [45]. The subthreshold and gate leakage current are the principal contributors to the total leakage current, resulting in leakage power dissipation. As the size of SRAM memory increases, the leakage power also increases due to an increase in memory cells. Therefore, determining the ETB size involves a trade-off between extending the lifetime and minimizing static power consumption. Based on our experimental analysis, we set the ETB size to eight blocks (512B) for our experiments. During execution, when recently written blocks fill the ETB, existing blocks must be replaced to accommodate new ones. TANC identifies the least recently written block among

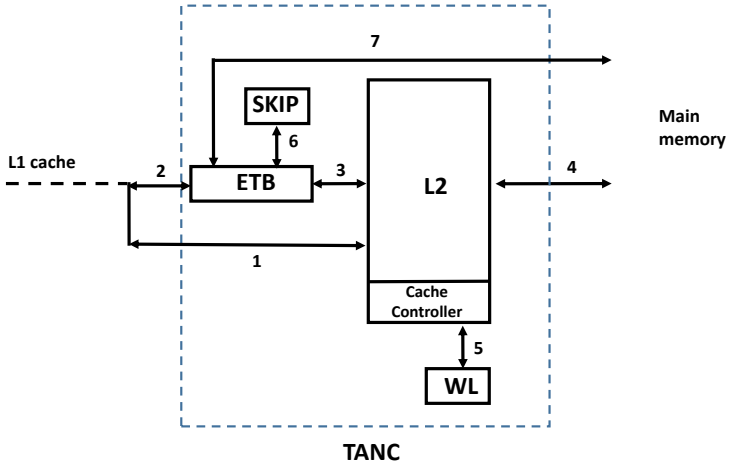

Figure 6.3: Schematic representation of TANC enabled L2 cache

the ETB entries as the victim block, using a three-bit counter associated with each entry. A counter value of zero indicates the block with the oldest write history. The wear-leveling module is active only for specific TANC variants, enhancing write distribution through an inter-pair way swapping mechanism similar to the state-of-the-art ENDURA [46] technique. The skip module is key component of the TANC_ALL_SKIP variant, which is activated when ETB write-backs are frequent to the L2 cache due to a lack of memory locality. We summarize the key features of various TANC variants.

- Enhances lifetime of MLC NVM LLCs by utilizing unused ETBs for write reduction.
- Improves the write energy and latency by redirecting frequent writes to SRAM based ETB.
- Reduces the intra-set variation using a naive *SpH* wear-leveling policy.
- Addresses the LLC thrashing issue in applications with poor temporal locality.

Utilizing soft-ways, which offer quicker access and lower write energy consumption than hard-ways, to store heavily written blocks can enhance MLC NVM cache write

Table 6.1: Service of requests in TANC

Request type	Request	ETB	L2	Action	Sequence of access
1	Read	Miss	Miss	Main Memory Access	Request : 1||2 -> 4 Response : 4 -> 1
2	Read	Miss	Hit	Service the request from L2	Request : 1||2 Response : 1
3	Read	Hit	Miss	INVALID STATE	Not Applicable
4	Read	Hit	Hit	Service the request from ETB	Request : 1||2 Response : 2
5	Write	Miss	Miss	Main Memory Access	Request : 1||2 -> 4 Response : 4 -> 1
6	Write	Miss	Hit	Write to ETB	Request : 1||2 Response : 3 ->2
7	Write	Hit	Miss	INVALID STATE	Not Applicable
8	Write	Hit	Hit	Service the request from ETB	Request : 1||2 Response : 2

latency and energy efficiency. However, this strategy leads to uneven write distribution and premature degradation of soft-ways, negatively impacting the overall lifespan of the NVM cache. TANC addresses this issue by utilizing the unused SRAM-based ETB to handle write requests for MLC NVMs. Leveraging the faster and more energy-efficient write capabilities of SRAM cells, our proposed method reduces latency and write energy while extending the relative lifetime of MLC NVM caches. Unlike hybrid caches, which require more space and suffer from leakage power issues, TANC optimally utilizes NVM features and allocates unused resources like the ETB to enhance performance. Various types of potential requests within TANC are outlined in Table 6.1. The columns of Table 6.1 shows request categories, request characteristics, presence in either the ETB or L2 cache, actions taken for each request, and their control flow as depicted in Figure 6.3. The L2 cache controller examines the ETB and L2 for every request. Given that ETBs function as SRAM-based storage units linked with processors, all read-write requests are handled within a single cycle. Request types 1 and 5 denote read and write requests, respectively, which experience misses in both the ETB and L2, necessitating their forwarding to main memory as standard cache miss requests. Type 2 requests signify read accesses for blocks missing in the ETB but present in the L2, which TANC accommodates by serving them directly from the L2 cache. Since blocks in the ETB are invariably written from the L2 cache, an L2 cache miss leads to an ETB miss, making type 3 and 7 requests nonexistent. Read and write requests that hit the ETB (request types 4 and 8) are catered to directly from the ETB. In instances of L2 write hits which are absent in the ETB (request type 6), the block is written to the ETB, and subsequent requests

Table 6.2: TANC variants and associated modules

Variant Name	ETB contents	Wear-leveling	Skip
TANC_HW	Hard-way blocks only	Inactive	Inactive
TANC_HW_SW	Hard-way and selected soft-way blocks	Inactive	Inactive
TANC_ALL	Hard-way and soft-way blocks	Inactive	Inactive
TANC_HW_WL	Hard-way blocks only	Active	Inactive
TANC_HW_SW_WL	Hard-way and selected soft-way blocks	Active	Inactive
TANC_ALL	Hard-way and soft-way blocks	Active	Inactive
TANC_ALL_SKIP	Hard-way and soft-way blocks	Inactive	Active

are fulfilled from the ETB until eviction occurs.

Depending on the processor, ETBs come in sizes ranging from 8KB to 32KB [47]. Our experimental studies show that increasing the ETB size in TANC enhances the lifetime and performance of NVM LLC, but it amplifies the leakage power associated with the SRAM-based ETB. Based on empirical analysis, we fix ETB size as 512B (equivalent to 8 cache blocks) as the optimal design choice. Considering selection criteria for block placement in the ETB, wear-leveling techniques, and ETB write-back policies, we propose and evaluate seven variants of TANC, as summarized in Table 6.2. The primary objective across all TANC variants is to minimize L2 writes by utilizing the ETB. The initial three variants solely employ strategies for minimizing writes. The subsequent three variants incorporates an additional wear-leveling module to improve lifetime further. The final variant adopts a skip cache approach to mitigate the impact of frequent ETB write-backs. The following sections discuss these techniques comprehensively.

6.3.3 TANC variants with write minimization only

The main goal of TANC is to improve the lifetime and efficiency of MLC NVM by reducing write operations using ETB. This section discusses three different approaches that utilize ETB to reduce the frequency of writes to NVM, each varying in the selection of blocks to keep in ETB. Following sections briefly describe the specific techniques used in these approaches.

- TANC_HW:

  As previously discussed, hard-way writes are costly compared to soft-way writes. Each hard-way write requires an additional write to the corresponding soft-way, reducing the memory lifetime. Conversely, soft-way writes involve a single-step procedure. In our initial variation, referred to as TANC_HW, the ETB is exclusively designated for hard-way writes. When a write request is initiated to the L2 cache, TANC assesses whether the requested block, such as block A, is stored in a hard or soft way. If block A resides in a hard-way of the cache, the process is executed by writing it in the ETB. Subsequent requests to block A, until its eviction will be serviced from the ETB. In case of a hit in a soft-way, the corresponding operation is managed directly from the L2 cache. Cache misses are handled conventionally. The ETB is exclusive for storing hard-way blocks.

- TANC_HW_SW

  Frequent writes to the hard-ways and repeated writes to the soft-ways can adversely affect the lifetime of MLC NVM. Hence, it is important to prioritize heavily written soft-ways along with hard-ways for extending MLC NVMs lifetime. In the TANC_HW_SW approach, the ETB accommodates hard-ways and some frequently accessed soft-ways for write operations. When a soft-way undergoes consecutive writes n times, it gains the status of a hard-way and is stored in the ETB. During the ETB replacement block selection, hard-way blocks are selected if and only if soft-way blocks absent in the ETB.

- TANC_ALL

  As discussed, multiple soft-way writes are as important as hard-way writes. Although the first two variants partly address challenges linked with hard-way and repeated soft-way writes, some scenarios necessitate more frequent access to soft-ways over hard-way writes. To address this issue, we present the third variation of TANC, called TANC_ALL. In this version, all write requests are managed via ETB, irrespective of their source, by directing them to ETB for the initial write process.

6.3.4 TANC Variants With Write Minimization And Wear-leveling

Write variation occurs when certain segments of memory are accessed more frequently for write operations than others. This discrepancy in write frequency can occur due to various factors such as irregular write patterns in applications or intentional repetitive writes induced by malicious attacks. Such write variations is visible at various levels within the memory hierarchy. As mentioned in the previous chapters, wear-leveling strategies aim to distribute writes evenly, thereby minimizing write variation. In the context of cache memory, write variation is categorized into intra-set variation, occurring within a set, and inter-set variation, which occurs across sets [6]. These variations are quantified using the coefficients of intra-set ($IntraV$) and inter-set ($InterV$) variation, respectively. If the write distribution is uniform, each cache way will have approximately equal write counts, resulting in both $InterV$ and $IntraV$ values approaching zero. A notable variation in inter-set writes suggests that cache lines in different sets may experience significantly different write frequencies due to biased address residency in applications. Similarly, substantial intra-set write variations occur when a single cache line in a set receives frequent cache write hits, causing it to absorb a significant portion of the cache writes. Consequently, the remaining $M-1$ lines in the set (for an M-way associative cache) may have unevenly distributed write accesses. Maintaining low values for $InterV$ and $IntraV$ helps in mitigating the risk of early wear-out of memory cells. Various advanced techniques for SLC NVM and MLC NVM rely on uniform write distribution. TANC_HW_WL, TANC_HW_SW_WL, and TANC_ALL_WL are variants that integrate wear-leveling into the base TANC model. ETB reduces the actual number of writes to NVM but does not guarantee uniform distribution across the memory. Wear-leveling techniques such as ENDURA [46] employ customized strategies for MLC NVM-based caches. We incorporate ENDURA's SpH wear-leveling strategy into TANC variants to ensure more uniform write distribution across the cache memory. The wear-leveling module, depicted in Figure 6.2, gets activated after a fixed time interval (100K cycles). We utilize a three-bit saturation counter per hard-way/soft-way pair to estimate block write counts. While the wear-leveling module is active, write-back operations from

ETB to the L2 cache trigger way pair swapping. This swapping exchanges the pair with the highest write count and one with the lowest within a set, thereby improving write distribution. Swapping is aborted if all counters are saturated, indicating heavy writes to all pairs. After swapping, the counters are reset. Since ETB ensures that most repeated write requests bypass the L2 cache, we opt for a less accurate estimate of write counts, significantly reducing associated overhead. Activation of the wear-leveling module by an ETB write-back at fixed intervals ensures that swapping and related writes do not degrade the L2 cache's lifetime. We analyze the impact of the wear-leveling module in TANC in the experimental setup and results analysis section.

6.3.5 TANC with Skip cache

Cache memories become effective only when applications exhibit either spatial or temporal locality. However, specific applications fail to demonstrate locality either throughout their entire execution or during specific phases. Consequently, such applications incur a higher cache miss rate, leading to a phenomenon known as thrashing [48]. Notably, applications like *bzip2* and *lbm* exhibit a high miss rate, reducing the impact of techniques like TANC. This degradation in effectiveness is resulted from the absence of locality, resulting in frequent cache evictions from the ETB. To mitigate this issue, we propose modifying the existing skip cache approach [49]. In this approach the thrashing and non-thrashing phases of application execution are identified and bypasses the cache accordingly using shadow tags to monitor application behaviour. Specifically, the application execution is segmented into 50-million-cycle phases to distinguish thrashing from non-thrashing phases. A variant of TANC, namely TANC_ALL_SKIP, tackles the memory thrashing problem by leveraging ETB writes to identify thrashing phases of application execution. Skip cache-enabled TANC monitors ETB misses and activates skip cache mode upon surpassing a predetermined threshold. During skip cache mode, write-backs

ALGORITHM 6: Working of TANC_ALL_SKIP technique

```
1  ETB_HIT : TRUE if requested block is present in ETB else FALSE;
2  L2_HIT :TRUE if requested block is present in L2 else FALSE;
3  ETB_MISS_COUNT =0 : Counts consecutive ETB write miss;
4  ETB_FULL = FALSE :TRUE if ETB is full else FALSE;
5  SKIP = FALSE : FALSE if skip inactive else TRUE;
6  MAX = 10 : Maximum number of consecutive ETB write miss for activate skip cache;
7  ETB_READ : ETB READ HIT; Requested block is read from ETB;
8  ETB_UPDATE : ETB WRITE HIT; Requested block is updated in ETB;
9  L2_READ : L2 READ HIT; Requested block is read from L2;
10 ETB_WRITE : L2 WRITE HIT; Requested block is written to ETB;
11 MM_ACCESS : L2 Miss; Access Main Memory for the corresponding request;
12 ETB_EVICT_L2 : Write back the LRU block to L2;
13 ETB_EVICT_MM : Write back the LRU block to Main Memory;
14 repeatfor every L2 cache request R and block B do
15     if R==read then
16         if ETB_HIT then
17             ETB_READ;
18         else
19             if L2_HIT then
20                 L2_READ;
21             else
22                 MM_ACCESS;
23     else
24         if ETB_HIT then
25             ETB_UPDATE;
26             ETB_MISS_COUNT=0;
27         else
28             ETB_MISS_COUNT++;
29             if L2_HIT then
30                 if ETB_FULL;
31                  then
32                     if SKIP then
33                         ETB_EVICT_MM;
34                         ETB_WRITE;
35                     else
36                         ETB_EVICT_L2;
37                         ETB_WRITE;
38                 else
39                     ETB_WRITE;
40             else
41                 MM_ACCESS;
               if ETB_MISS_COUNT >=MAX then SKIP=TRUE;
```

from ETB are directed to the next level of the memory hierarchy (main memory) rather than the corresponding L2 cache. Activation of skip mode is triggered by ten consecutive write misses to the trace buffer, and the system reverts to the default mode immediately after the first trace buffer hit. Detailed working of the skip cache module are provided in Algorithm 6. Further discussion on the experimental results and analysis of this technique is presented in the experimental results and analysis section.

6.4 Experimental Setup and Result Analysis

We use gem5 simulator for modeling our proposed variants of TANC architectures using the configuration given in Table 6.3. Similar to the previous chapters we evaluate the performance of our proposed techniques by we categorizing the SPEC CPU2006 benchmark programs into three categories based on WPKI and MPKI for L2 cache and as summarized in Table 3.1. We do sufficient fast-forwarding and execute at least one billion instructions covering the benchmarks sim points to collect statistics for further analysis. We use Immediate Restore Scheme (IRS) [23] to address the write disturbance where for every hard-way write, the corresponding soft-way is read first and restored immediately after the completion of the hard-way write operation. We use the RUBY memory model and MESI Two-Level protocol to maintain cache coherence. To do a fair analysis, we compare TANC and its variants given in Table 6.2 with the following conventional NVM architectures denoted by,

- SLC_C: SLC NVM based L2 cache without any optimization
- SLC_ViSC: SLC NVM based L2 cache employing Virtually Split Cache technique [50]
- MLC_C: MLC NVM based L2 cache without any optimization
- MLC_WL: MLC NVM based L2 cache with the naive *SpH* wear-leveling policy [46]
- ENDURA: A state-of-the-art wear-leveling technique customized for MLC NVM based L2 cache[46]

Table 6.3: Simulation parameters

CPU	Unicore, 2 GHz
L1 Cache	Private, 32KB, SRAM Split cache, 4-way set associative, 64B block
L2 Cache	4MB STT-RAM, 8-way set associative, 64B block
Main Memory	8 GB
ETB	512B, SRAM
Wear-leveling interval	100000 Cycles

- RESTORE: A state-of-the-art performance and lifetime improvement technique customized for MLC NVM based L2 cache [24]

In subsequent sections, we discuss the impact of TANC on various parameters compared to other state-of-the-art lifetime improvement technique based on different popular performance metrics.

6.4.1 Impact on Relative Lifetime

The lifetime of NVM cells is a critical factor, especially considering their susceptibility to premature aging due to limited write endurance. Exploiting security vulnerabilities, malicious users may employ repeated write attacks to exploit these limitations. MLC NVMs, with their multi-step writing process, are particularly vulnerable to early wear out compared to SLC NVMs. Like previous chapters, we determine the effectiveness of our proposed approach in improving the lifetime using relative lifetime. Figure 6.4 illustrates the geometric mean of relative lifetime across various benchmarks, comparing TANC variants and other state-of-the-art wear-leveling techniques for SLC and MLC NVMs. SLC_ViSC enhances lifetime by partitioning the unified cache into virtual instruction and data caches, redistributing concentrated writes across memory. ENDURA employs inter-pair swapping to balance write distribution in MLC NVMs, enhancing lifetime. RESTORE minimizes restoration writes by deactivating soft-ways associated with frequently used hard-ways. Implementing wear-leveling with TANC ensures even write distribution in the cache, further enhancing lifetime. The skip cache module addresses poor cache locality by diverting ETB evictions to main memory when thrashing occurs, thereby reducing L2 cache wear. Results, normalized to SLC_C, show SLC_ViSC improving lifetime by 66%, highlighting the

efficacy of virtual cache partitioning. MLC_C exhibits approximately two-thirds of SLC_C's lifetime due to its multi-step writing policy, with MLC_WL enhancing its lifetime by 19%. ENDURA and RESTORE show 3% and 14% lifetime improvements over SLC_C, respectively, underlining the effectiveness of their respective strategies.

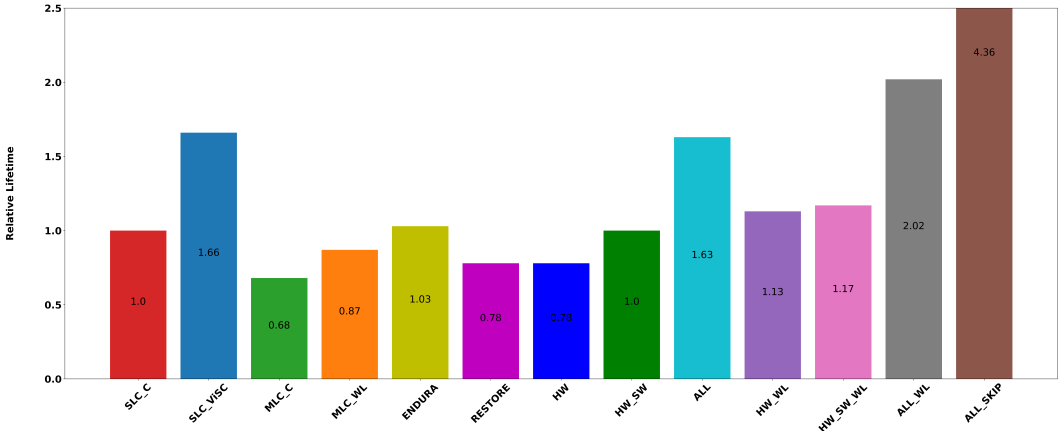

Figure 6.4: Relative lifetime of various techniques

All versions of TANC utilize ETB to limit writes to the L2 cache, significantly enhancing their relative lifetime. The effectiveness of this technique depends largely on the ETB access policy and application patterns. TANC_HW, which exclusively employs ETB for hard-way writes, shows a 10% enhancement over SLC_C. Particularly, TANC_HW excels in benchmarks such as *namd*, characterized by low WPKI and high write variation. Each hard-way write triggers an associated soft-way write during restoration operations. TANC_HW mitigates such redundant writes, especially in repeated writes to hard-ways. Variants of TANC without wear-leveling reduce L2 cache writes but do not ensure uniform distribution. TANC_HW_WL, a basic wear-leveling policy applied to TANC_HW, boosts relative lifespan by 35% owing to its efficient write distribution. By utilizing ETB to process write requests to frequently written soft-ways instead of hard-ways, TANC_HW_SW outperforms TANC_HW, achieving a mean lifetime similar to SLC_C. Furthermore, wear-leveling (TANC_HW_SW_WL) enhances its lifetime by an additional 17%. TANC_ALL, employing ETB for all write requests, exhibits superior performance due to minimized L2 writes. It exclusively handles write requests from ETB, with L2 cache writes limited to write backs from ETB. TANC_ALL's mean relative lifetime matches that

of SLC_ViSC, with wear-leveling further extending it by 39%. Enabling cache skipping for TANC_ALL yields the most substantial lifetime improvement, particularly beneficial for benchmarks like *bzip2, lbm* and *libquantum*, which suffer from significant cache thrashing due to frequent writebacks from ETB. TANC_ALL enhances MLC_C's mean lifespan by 4.36x compared to SLC_C and 2.62x to SLC_ViSC.

Table 6.4: Comparison of relative lifetime of various techniques for different benchmark applications

	Low WPKI				Mid WPKI			High WPKI			Mean
	Gr	H2	Nd	So	Bz	Lq	Sj	Hm	Lb	Ls	
SLC	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
SLC_ViSC [50]	1.03	2.75	3.44	1.76	1.00	1.00	3.16	2.22	1.26	1.08	1.66
MLC	0.93	0.57	0.81	0.74	0.52	0.50	0.74	0.88	0.69	0.56	0.68
MLC_WL	1.38	1.25	1.12	0.69	0.52	0.50	1.13	1.17	0.78	0.73	0.87
MLC_ENDURA [46]	1.30	1.58	2.85	0.81	0.52	0.51	1.67	1.40	0.76	0.57	1.03
MLC_RESTORE [24]	0.93	0.62	0.81	0.74	0.52	0.52	0.74	0.88	0.69	2.01	0.78
TANC_HW	0.93	0.99	1.00	0.76	0.52	0.50	0.77	0.99	0.70	0.90	0.78
TANC_HW_SW	1.41	1.00	6.91	0.76	0.52	0.50	0.77	1.00	0.70	1.01	1.00
TANC_ALL	1.41	1.74	53.58	0.86	0.52	0.50	0.86	4.01	0.75	1.72	**1.63**
TANC_HW_WL	1.42	1.67	2.51	0.85	0.52	0.50	1.59	1.74	0.78	1.23	1.13
TANC_HW_SW_WL	1.42	1.81	3.40	0.85	0.52	0.50	1.59	1.76	0.78	1.13	1.17
TANC_ALL_WL	1.26	2.53	77.79	0.92	0.53	0.50	1.60	6.50	0.83	2.20	**2.02**
TANC_ALL_SKIP	3.27	2.14	53.80	1.37	1.94	114.44	1.45	4.01	2.5	2.50	**4.36**

Table 6.4 shows the relative lifetime of various techniques across different benchmarks. The best values among the variants are highlighted in bold. In the low WPKI benchmark category, particularly those with a high L2 cache hit rate, TANC_ALL variants exhibit notable performance enhancements. The elevated L2 hit rate corresponds to a higher ETB hit rate, reducing writes to the L2 cache and enhancing the lifetime. For benchmarks like *namd* and *h264ref*, where most write requests are ETB hits and SKIP mode is rarely activated, TANC_ALL and TANC_ALL_WL outperform TANC_ALL_SKIP. Specifically, *namd* experiences significant lifetime improvement with TANC_ALL and TANC_ALL_WL due to its high L2 hit rate and predominant soft-way writes. Conversely, the lower hit rate of *soplex* minimizes the impact of ETB, resulting in the superior performance of TANC_ALL_SKIP compared to other TANC variants. In mid-WPKI benchmarks such as *bzip2* and *libquantum*, MLC NVM caches show better lifetime when TANC_ALL_SKIP is

activated. For instance, in *libquantum* with a meager L2 hit rate, TANC_ALL_SKIP significantly prolongs the lifetime by mitigating thrashing effects, whereas other techniques offer marginal improvements. Wear-leveling strategies like ViSC for SLC NVM, ENDURA for MLC NVM, and the naive *SpH* technique in other variants have minimal impact due to the negligible influence of write distribution schemes on such applications with low L2 hit rates. These applications trigger frequent evictions from ETB thus TANC exhibits minimal impact on lifetime. The skip cache policy of writing back to memory instead of L2 cache improves lifetime as recently accessed blocks are not re-referenced. Among mid-WPKI benchmarks, *sjeng* with the highest L2 hit rate shows superior performance with TANC_ALL compared to TANC_ALL_SKIP, a trend also observed in high WPKI benchmarks. For instance, *hmmer* benefits from a high L2 hit rate, enabling ETB to significantly reduce L2 writes and thereby improve lifetime, whereas *lbm* and *leslie3d*, with their low hit rates, achieve better lifetime improvement by skipping cache for ETB writeback.

6.4.2 Impact on Average Memory Access Time

The high write latency of MLC NVM is a serious concern when utilized as cache memories. Figure 6.5 illustrates the normalized geometric mean of average memory access time in cycles (AMAT) across various cache architectures. For a given cache size, the write latency of hard-ways is twice that of soft-ways and SLC NVMs, and 5x greater than SRAM. SLC_C and SLC_ViSC exhibit similar average memory access times, as ViSC employs wear-leveling techniques that do not minimize or bypass L2 writes. However, the write latency of MLC_C varies depending on the location of the requested cache block. Due to the high write latency of hard-ways, MLC_C and MLC_WL demonstrate a 6% higher AMAT than SLC_C. ENDURA employs intra-pair swapping to ensure frequently accessed blocks are written in soft-ways, resulting in a 7% reduction in the mean AMAT of MLC NVMs. Despite better application write patterns, RESTORE exhibits a similar AMAT to MLC_C. With the majority of writes handled by SRAM-based ETB, TANC variants notably reduce L2 write latency, which is reflected in their AMAT values. TANC_HW and TANC_HW_SW exhibit similar AMAT to ENDURA. In the TANC_ALL variant, where ETB manages all

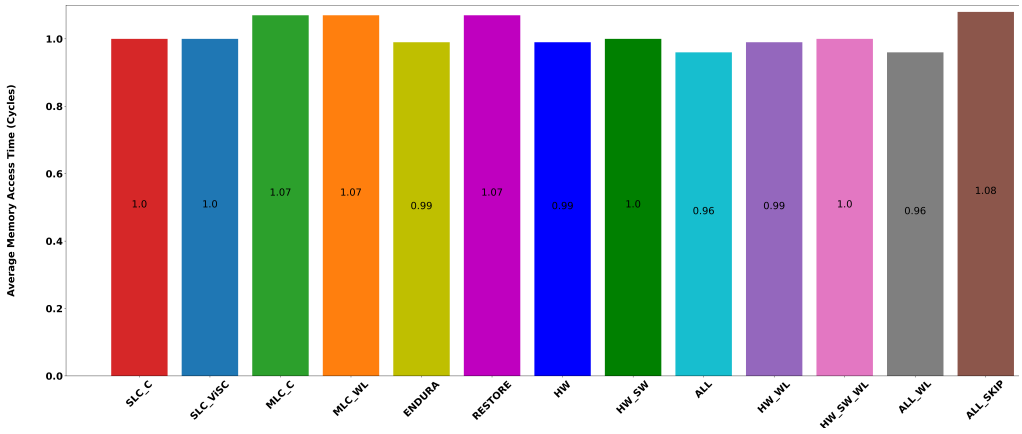

Figure 6.5: Comparison of normalized AMAT (cycles) of various techniques

write accesses, there is an 11% decrease in AMAT compared to MLC_C. Conversely, the TANC_ALL_SKIP variant writes back to the main memory instead of the L2 cache during the active skip mode phase, resulting in a 1% increase in mean AMAT. Table 6.5 shows the average memory access time (AMAT) corresponding to each cache architecture across various benchmark applications. Lowest values are highlighted in bold for easy reference. As the number of write hits in the ETB increases, the AMAT decreases due to faster write accesses to the SRAM-based ETB than the MLC NVM L2 cache. For applications falling within the low WPKI category, such as *namd*, which exhibits a high cache hit rate and minimal memory thrashing, skip mode remains inactive for the majority of the execution duration, resulting in a reduced AMAT. Conversely, despite *gromacs* with a high L2 cache hit rate, the activation of skip mode during execution leads to increased writebacks from the ETB to main memory, consequently increasing the average access time for TANC_ALL_SKIP. Similarly, for mid and high-category WPKI benchmarks, such as *hmmer*, which have a high cache hit rate, TANC_ALL_SKIP similar to the TANC_ALL variant as skip mode remains rarely active during execution. The *SpH* naive wear-leveling technique, associated with various TANC variants, facilitates swapping heavily written and lightly written hard-way soft-way pairs. Unlike in ENDURA which also facilitates intra-pair swapping between hard-way and soft-way, our *SpH* naive wear-leveling policy minimally impacts L2 write latency and AMAT.

Table 6.5: Comparison of normalized AMAT (cycles) of various techniques for different benchmark applications

	Low WPKI				Mid WPKI			High WPKI			Mean
	Gr	H2	Nd	So	Bz	Lq	Sj	Hm	Lb	Ls	
SLC	1.97	1.07	1.05	1.05	2.14	1.72	1.9	1.96	3.6	2.12	1.73
SLC_ViSC [50]	1.97	1.07	1.05	1.05	2.14	1.72	1.9	1.96	3.6	2.12	1.73
MLC	2.32	1.09	1.06	1.07	2.27	1.77	2.01	2.32	3.86	2.29	1.85
MLC_WL	2.32	1.09	1.06	1.07	2.27	1.77	2.01	2.32	3.86	2.29	1.85
MLC_ENDURA [46]	1.83	1.07	1.04	1.06	2.13	1.71	1.93	1.78	3.61	2.17	1.72
MLC_RESTORE [24]	2.25	1.09	1.05	1.07	2.27	1.77	2.01	2.31	3.86	2.29	1.86
TANC_HW	1.75	1.08	1.04	1.06	2.21	1.73	1.97	1.75	3.76	2.14	1.72
TANC_HW_SW	1.75	1.08	1.03	1.06	2.24	1.73	1.97	1.75	3.76	2.25	1.73
TANC_ALL	1.57	1.08	1.03	1.06	2.14	2.09	1.9	1.57	3.59	1.71	**1.65**
TANC_HW_WL	1.75	1.08	1.04	1.06	2.21	2.14	1.97	1.75	3.76	1.73	1.72
TANC_HW_SW_WL	1.75	1.08	1.03	1.06	2.24	2.25	1.97	1.75	3.76	1.73	1.73
TANC_ALL_WL	1.57	1.08	1.03	1.06	2.14	2.09	1.9	1.57	3.59	1.71	**1.65**
TANC_ALL_SKIP	2.67	1.11	1.03	1.08	2.47	2.3	2.16	1.57	4.3	1.87	1.87

6.4.3 Impact on Write Energy

The high energy write energy associated with MLC NVMs is serious challenge in implementing L2 caches. Compared to soft-ways and SLC NVMs, hard-ways incur 3x more energy for write operations and are 16x more energy intensive than SRAM for a given cache size. Our proposed techniques aim to mitigate this issue by reducing the average write energy of MLC NVM LLCs. TANC shows a significant decrease in average write energy similar to that of SLC NVMs. Figure 6.6 shows the average write energy of the proposed techniques alongside MLC_C, SLC_C, and state-of-the-art wear leveling techniques for both SLC and MLC NVMs. In ENDURA, intra-pair swapping, which prioritizes frequently written blocks to soft-ways, contributes to an 8.6% reduction in the mean write energy. By diverting the high energy hard-way writes to ETB in TANC_HW and TANC_HW_SW, energy consumption is reduced by 30% and 31.5%, respectively. TANC_HW outperforms TANC_HW_SW due to the latter's sharing of ETB for selected soft-way blocks, necessitating the writing back of some hard-way blocks to the L2 cache, increasing the mean L2 write energy. In TANC_ALL, where all writes are serviced by ETB resulting in lower write energy hence the mean L2 write energy is reduced by 47.2%, surpassing that of SLC_C.

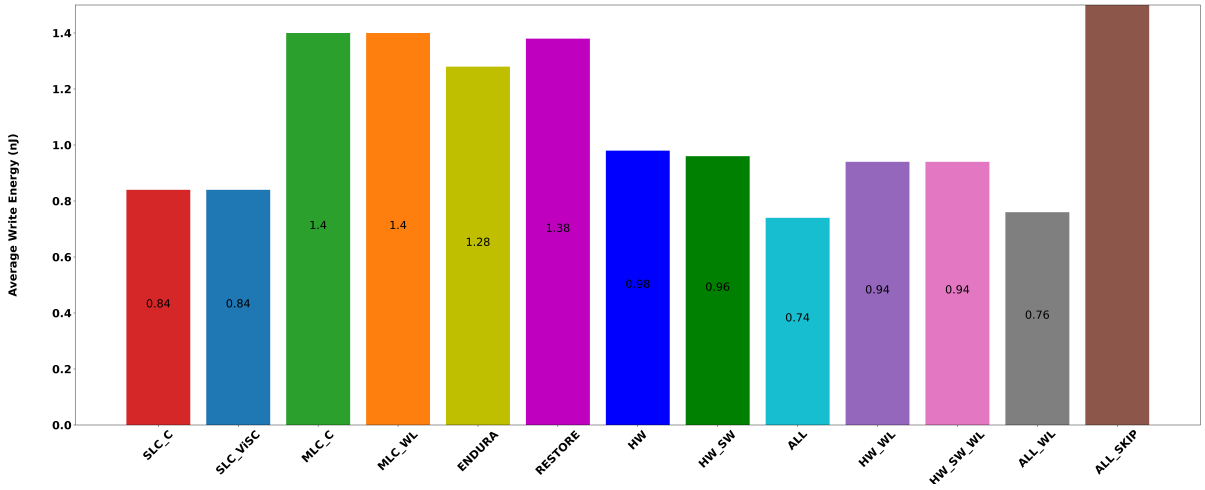

Figure 6.6: Comparison of average write energy (nJ) of various techniques (shorter the bar, the better)

However, during skip mode, TANC_ALL_SKIP writes back the blocks evicted from ETB to the main memory, resulting in higher mean write energy than other modes. Table 6.6 presents the write energy of the proposed techniques for each benchmark. Lowest values for write energy are highlighted in bold. Notably, for benchmarks such as *namd* and *hmmer*, which exhibit minimal L2 thrashing, TANC_ALL_SKIP show less write energy consumption.

Similar to AMAT, the average write energy of L2 caches varies depending on the application. Among all benchmark categories, TANC_ALL exhibits the lowest average write energy. This is attributed to bypassing of all writes by TANC_ALL to SRAM-based ETB which possesses significantly lower write energy. However, when ETB writebacks to DRAM-based main memory occur, particularly during skip mode in TANC_ALL_SKIP, the mean write energy increases due to the high energy consumption of DRAM writes. Applications like *namd* and *hmmer* show lower write energy because they seldom activate skip mode due to their access patterns, resulting in fewer ETB to main memory writebacks

6.4.4 Impact on Hard-way Writes

Table 6.7 shows the average reduction of hard-way writes for different TANC variants.

Table 6.6: Comparison of average write energy (nJ) of various techniques for different benchmark applications

	Low WPKI				Mid WPKI			High WPKI			Mean
	Gr	H2	Nd	So	Bz	Lq	Sj	Hm	Lb	Ls	
SLC	0.84	0.84	0.84	0.84	0.84	0.84	0.84	0.84	0.84	0.84	0.84
SLC_ViSC [50]	0.84	0.84	0.84	0.84	0.84	0.84	0.84	0.84	0.84	0.84	0.84
MLC	1.4	1.38	1.42	1.4	1.4	1.39	1.39	1.4	1.4	1.4	1.40
MLC_WL	1.4	1.39	1.44	1.4	1.4	1.4	1.4	1.4	1.4	1.4	1.40
MLC_ENDURA [46]	1.23	1.13	1.28	1.4	1.38	1.39	1.39	0.9	1.46	1.4	1.28
MLC_RESTORE [24]	1.4	1.38	1.30	1.4	1.4	1.4	1.39	1.39	1.4	1.36	1.38
TANC_HW	1.1	1.04	0.59	1.08	1.19	0.95	1.19	0.64	1.2	1.09	0.98
TANC_HW_SW	1.1	1.04	0.39	1.08	1.19	1.12	1.19	0.65	1.2	1.09	0.96
TANC_ALL	0.9	0.86	0.35	0.9	0.9	0.8	0.9	0.4	0.9	0.9	**0.74**
TANC_HW_WL	1.09	1.05	0.63	1.09	1.09	0.8	1.08	0.61	1.09	1.09	0.94
TANC_HW_SW_WL	1.09	1.05	0.56	1.09	1.09	0.95	1.08	0.61	1.09	1.09	0.94
TANC_ALL_WL	0.9	0.87	0.38	0.91	0.9	0.95	0.9	0.4	0.9	0.9	**0.76**
TANC_ALL_SKIP	21.03	17.86	2.93	21.8	24.66	12.99	22.43	0.48	25.36	25.67	11.83

Table 6.7: Average reduction of hard-way writes for TANC variants

	Reduction in hard-way writes
TANC_HW	23.08%
TANC_HW_SW	23.05%
TANC_ALL	21.13%
TANC_HW_WL	22.50%
TANC_HW_SW_WL	22.48%
TANC_ALL_WL	20.35%
TANC_ALL_SKIP	88.89%

TANC_ALL_SKIP minimizes the occurrence of two-step writes (hard-way writes) by redirecting all write operations to the ETB and in skip mode, transfers data directly to main memory instead of the L2 cache. Among the different variants, TANC_HW demonstrates the most significant reduction in hard-way writes by exclusively allocating the ETB for this purpose. Table 6.7 shows that wear-leveling variants marginally elevate the count of hard-way writes compared to their base variant. This slight increase in hard-way writes primarily due to the wear-leveling process and the additional writes resulting from inter-pair swapping.

6.4.5 Impact on Intra-set and Inter-set variations

Due to the multi-step writing process inherent in MLC NVMs, the write variation in MLC NVM caches is notably higher compared to SLC NVM caches. The core concept of TANC is minimizing write accesses to the L2 cache through ETB and does not directly impact on intra-set and inter-set variations. Figure 6.7 shows the comparison of the $IntraV(\%)$ value across different TANC variants, as calculated using equation 3.1, while Table 6.8 presents the same analysis for individual applications. Best case values are highlighted in bold fond for easy analysis. For most of mid and high category benchmarks like *bzip2* and *lbm*, TANC has negligible impact on write variation. However, benchmark programs with low WPKI exhibit an increase in variation. TANC's approach focuses on reducing the total number of writes rather than concentrating them, resulting in significant differences between average and maximum writes in cache memory, leading to substantial intra-set and inter-set write variations. Variants employing the naive SpH wearing policy alongside TANC demonstrate improved write distribution, yielding lower intra-set variation values. The proposed techniques employ intra-set wear leveling to distribute writes within the cache memory set. TANC_ALL_SKIP notably reduces writes to the cache, yet it does not affect the write distribution pattern, resulting in higher $IntraV$ values and negligible impact on inter-set variation.

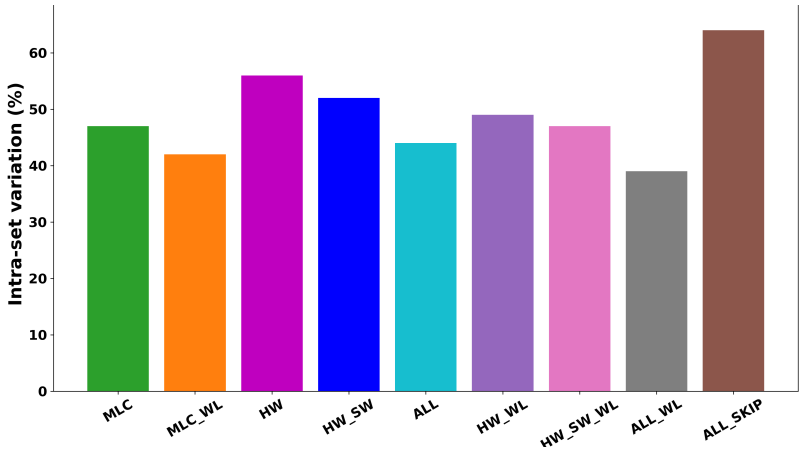

Figure 6.7: Comparison of intra-set write variation of various TANC variants (shorter the bar, the better)

Table 6.8: Comparison of intra-set write variation* of various TANC variants for different benchmark applications

	Low WPKI				Mid WPKI			High WPKI			Mean
	Gr	H2	Nd	So	Bz	Lq	Sj	Hm	Lb	Ls	
MLC	55	72	118	39	34	36	50	37	36	37	47
MLC_WL	45	54	70	38	36	36	40	36	36	36	**42**
TANC_HW	54	66	211	39	36	36	49	88	36	48	56
TANC_HW_SW	54	66	110	39	36	36	49	89	36	48	52
TANC_ALL	54	66	54	39	36	36	49	43	36	36	44
TANC_HW_WL	47	56	97	38	36	36	40	89	36	47	49
TANC_HW_SW_WL	46	56	90	38	36	36	40	89	36	36	47
TANC_ALL_WL	46	53	39	38	36	36	40	38	36	36	**39**
TANC_ALL_SKIP	53	103	54	56	48	90	154	43	68	37	64

*Low value of intra-set variation shows better distribution of writes within the set of a cache memory

6.4.6 Overhead Analysis

State-of-the-art wear leveling methods for both MLC and SLC NVMs come with significant overheads, resulting from the counters, swapping buffers, and similar components. Another popular strategy is to employ hybrid caches integrating NVM and SRAM memory cells. Hybrid caches tend to occupy more space and consume more power than their NVM only counterparts. In the basic versions of TANC (which lack wear leveling), write reduction relies only on ETB with the aid of the cache controller. Since ETBs remain unused after the debugging phase, they do not contribute to additional spatial overhead. The *SpH* wear-leveling technique employs a three-bit saturating counter for each block alongside four SRAM-based swapping buffers of 64B each to facilitate inter-pair swapping. For a 4MB cache, the naive *SpH* wear-leveling technique introduces a storage overhead of 0.15%, which is 33% lower than ENDURA, a customized wear-leveling policy for MLC-NVM caches. Further optimization of TANC's wear-leveling policy could reduce the associated storage overhead even more.

6.5 Conclusion and Future works

MLC NVMs are promising candidates for implementing LLCs due to their high packing density and low leakage power. Our proposed method, TANC, and its variations address several key drawbacks of MLC NVMs, such as limited lifetime, high write latency, and high energy consumption. Compared to SLC NVM caches, TANC enabled MLC NVM caches demonstrates significant advantages, including up to 4.36x lifetime, write access times, and reduced energy usage. Contrasting SLC NVMs, MLC NVMs occupy 1.84x less space and consume 2.62x less leakage energy. MLC based caches exhibit a remarkable 42x reduction in leakage power and require 7.20x less space than conventional SRAM caches. These favourable characteristics of MLC NVMs, along with enhancements through TANC, make them attractive for implementing large on-chip memories. However, using ETBs can lead to thrashing in applications with poor L2 cache hit rates. Our proposed approach addresses this concern with a skip cache policy. Additionally, MLC NVM caches

often face challenges with multi-step write operations and associated overheads. To mitigate write disturbance, our technique employs the Immediate Restore Scheme (IRS). Furthermore, innovative strategies like the Adaptive Restoration Scheme (ARS) can minimize energy overheads from write disturbances in MLC NVMs. A tailored policy designed to alleviate write disturbance, working in conjunction with TANC, can effectively reduce the write energy consumption of L2 caches.

CHAPTER

7

Conclusion and Future Work

Emerging NVM techniques such as STT-RAM, PCM, and ReRAM are promising candidates for meeting the demand for high on-chip memory when executing modern applications. As mentioned in Chapter 1, NVMs have excellent packing density and low leakage power compared to traditional DRAM and SRAM technologies. However, the high write energy, high latency and limited lifetime are critical challenges that must be addressed when NVMs are implemented as LLCs. This chapter summarizes the contributions of the thesis along with some of the future possible research directions.

7.1 Thesis Summary

The thesis addresses the limited lifetime of NVMs when implemented in LLCs as three different contributions. The first contribution of the thesis is WALL-NVC. WALL-NVC is a dual-stage wear-leveling mechanism for improving the lifetime of SLC NVM caches. WALL-NVC uses Least Recently Used Cold Block (LRU-CB), an NVM-friendly replacement policy and wear-leveling module, which works in tandem to delay the aging of NVM memory cells. WALL-NVC improves lifetime by 2.90x with respect to baseline and shows 1.16x and 1.18x improvement compared to EqualWrites

and EqualChance, respectively and significantly reduces the intra-set write variation up to 98.91%. For dual-core and quad-core systems, WALL-NVC improves lifetime upto 3.34x and 4.11x, reduces intra-set write variation L2 caches upto 90.11% and 94.12%, respectively

The thesis's second contribution discusses ViSC and its variants, which logically split the unified NVM LLC into data and instruction ways, which will only store the corresponding blocks. ViSC changes the logical mapping after a fixed time interval called reorganization interval, which forces the ways exclusively reserved for data (heavily written) to function as instruction ways (less written) and vice versa, resulting in uniform distribution of heavily written ways across the cache memory. E-ViSC and P-ViSC, variants of ViSC, improve the relative lifetime by 1.71x and 1.57x, respectively, whereas ViSC improves the relative lifetime by 1.64x compared to the baseline.

The thesis discusses TANC and its variants as its third contribution. MLC NVMs have better packaging density than SLC NVMs. However, we must address the significant challenges such as limited write endurance, read/write disturbances, high energy, and latency for write operations to make them suitable for use as LLCs in the memory hierarchy. TANC uses a portion of ETB, which is left unused after post-silicon validation. Since ETB is an existing resource, the overhead of TANC is very less. Compared to the SLC NVM cache, the TANC offers up to 4.36x the lifetime, faster access times, and low energy. Compared to SLC NVM, MLC NVM has 1.84x smaller space and 2.62x less leakage energy. MLC-based caches exhibit 42x less leakage power and 7.20x less space when compared to conventional SRAM caches.

7.2 Future Research Directions

All the contributions made in this thesis have been focused on general-purpose applications. However, the future of computing is shifting towards domain-specific, heterogeneous architectures tailored to specific workloads. Artificial Intelligence (AI) and Machine Learning (ML) applications, in particular, are gaining immense

popularity and are deeply integrated into our day-to-day lives. These applications often require high-performance computing devices with substantial on-chip and off-chip data storage capabilities, which makes them power-hungry.

On-chip NVM caches offer a promising solution to address the power challenges of such systems due to their negligible leakage power. Additionally, MLC NVM caches can provide more efficient data storage, enabling significant power savings when optimized to handle the specific needs of AI/ML workloads. These optimizations could also address challenges such as limited write endurance, thereby extending the applicability of NVM in high-demand scenarios.

Another important future research direction is investigating the impact of security vulnerabilities and potential attacks on conventional cache memories in NVMs. While the contributions in this thesis mitigate most write/read attacks on cache memories by ensuring a uniform distribution of writes, specific advanced attacks, as highlighted in the latest literature, can bypass wear-leveling techniques. A deeper analysis of such security vulnerabilities in NVMs and the development of robust countermeasures remains a promising area of exploration.

Furthermore, customized NVM caches designed explicitly for AI and ML applications could be transformative, significantly reducing the static power consumption in their compute nodes. Such advancements would enhance the energy efficiency of AI/ML systems and contribute to their sustainability, making them more viable for future large-scale deployments.

Bibliography

[1] S. Mittal and J. S. Vetter, "A survey of software techniques for using non-volatile memories for storage and main memory systems," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 27, no. 5, pp. 1537–1550, 2016.

[2] P. Chi, S. Li, Y. Cheng, Y. Lu, S. H. Kang, and Y. Xie, "Architecture design with stt-ram: Opportunities and challenges," in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2016, pp. 109–114.

[3] D. Gajaria, K. Antony Gomez, and T. Adegbija, "A study of stt-ram-based in-memory computing across the memory hierarchy," in *2022 IEEE 40th International Conference on Computer Design (ICCD)*, 2022, pp. 685–692.

[4] H.-S. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, "Phase change memory," *Proceedings of the IEEE*, vol. 98, no. 12, pp. 2201–2227, 2010.

[5] F. Zahoor, T. Z. Azni Zulkifli, and F. A. Khanday, "Resistive random access memory (RRAM): An overview of materials, switching mechanism, performance, multilevel cell (mlc) storage, modeling, and applications," *Nanoscale Res. Lett.*, vol. 15, no. 1, p. 90, Apr. 2020.

[6] 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)*, 2013, pp. 234–245.

[7] 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)*, 2009, pp. 14–23.

[8] C. Kim, J.-J. Kim, S. Mukhopadhyay, and K. Roy, "A forward body-biased-low-leakage sram cache: device and architecture considerations," in *Proceedings of the 2003 International Symposium on Low Power Electronics and Design, 2003. ISLPED '03.*, 2003, pp. 6–9.

[9] S. Segars, "Low power design techniques for microprocessors," 2000. [Online]. Available: https://api.semanticscholar.org/CorpusID:62153784

[10] S. Mittal and J. S. Vetter, "Ayush: A technique for extending lifetime of sram-nvm hybrid caches," *IEEE Computer Architecture Letters*, vol. 14, no. 2, pp. 115–118, 2015.

[11] ——, "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)*. USENIX Association, Oct. 2014.

[12] ——, "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, 2016.

[13] S. Agarwal and H. K. Kapoor, "Improving the lifetime of non-volatile cache by write restriction," *IEEE Transactions on Computers*, vol. 68, no. 9, pp. 1297–1312, 2019.

[14] ——, "Towards a better lifetime for non-volatile caches in chip multiprocessors," in *2017 30th International Conference on VLSI Design and 2017 16th International Conference on Embedded Systems (VLSID)*, 2017, pp. 29–34.

[15] 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, 2019.

[16] X. Bi, M. Mao, D. Wang, and H. Li, "Unleashing the potential of mlc stt-ram caches," in *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2013, pp. 429–436.

[17] A. Seznec, "A phase change memory as a secure main memory," *IEEE Computer Architecture Letters*, vol. 9, no. 1, pp. 5–8, 2010.

[18] M. K. Qureshi, A. Seznec, L. A. Lastras, and M. M. Franceschini, "Practical and secure pcm systems by online detection of malicious write streams," in *2011 IEEE 17th International Symposium on High Performance Computer Architecture*, 2011, pp. 478–489.

[19] R. Xu, E. H.-M. Sha, Q. Zhuge, Y. Song, and J. Lin, "Optimal loop tiling for minimizing write operations on nvms with complete memory latency hiding," in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2022, pp. 389–394.

[20] W. Dong, X. Li, Y. Li, M. Qiu, L. Dou, L. Ju, and Z. Jia, "Minimizing update bits of nvm-based main memory using bit flipping and cyclic shifting," in *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*, 2015, pp. 290–295.

[21] J. Li, L. Shi, C. J. Xue, C. Yang, and Y. Xu, "Exploiting set-level write non-uniformity for energy-efficient nvm-based hybrid cache," in *2011 9th IEEE Symposium on Embedded Systems for Real-Time Multimedia*, 2011, pp. 19–28.

[22] J.-W. Hsieh, Y.-T. Hou, and T.-C. Chang, "Alternative encoding: A two-step transition reduction scheme for mlc stt-ram cache," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 8, pp. 2753–2757, 2022.

[23] X. Chen, N. Khoshavi, R. F. DeMara, J. Wang, D. Huang, W. Wen, and Y. Chen, "Energy-aware adaptive restore schemes for mlc stt-ram cache," *IEEE Transactions on Computers*, vol. 66, no. 5, pp. 786–798, 2017.

[24] M. A. Qureshi, H. Kim, and S. Kim, "A restore-free mode for mlc stt-ram caches," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 6, pp. 1465–1469, 2019.

[25] P. Saraf and M. Mutyam, "Endurance enhancement of write-optimized stt-ram caches," in *Proceedings of the International Symposium on Memory Systems*, ser. MEMSYS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 101–113. [Online]. Available: https://doi.org/10.1145/3357526.3357538

[26] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 Simulator," *SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[27] 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, 2006.

[28] 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, p. 92–99, nov 2005. [Online]. Available: https://doi.org/10.1145/1105734.1105747

[29] J. Lowe-Power. (2024, January) Mesi_two_level. [Online]. Available: https://www.gem5.org/documentation/general_docs/ruby/MESI_Two_Level/

[30] J. L. Henning, "Spec cpu2006 benchmark descriptions," *SIGARCH Comput. Archit. News*, vol. 34, no. 4, p. 1–17, Sep. 2006. [Online]. Available: https://doi.org/10.1145/1186736.1186737

[31] 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, May 2013. [Online]. Available: https://doi.org/10.1145/2463585.2463592

[32] 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: Association for Computing Machinery, 2014, p. 139–144. [Online]. Available: https://doi.org/10.1145/2591513.2591525

[33] A.-T. Tran, D. S. Lakew, T.-V. Nguyen, V.-D. Tuong, T. P. Truong, N.-N. Dao, and S. Cho, "Hit ratio and latency optimization for caching systems: A survey," in *2021 International Conference on Information Networking (ICOIN)*, 2021, pp. 577–581.

[34] Y. Niranjan, S. Tiwari, and R. Gupta, "Average memory access time reduction in multilevel cache of proxy server," in *2013 3rd IEEE International Advance Computing Conference (IACC)*, 2013, pp. 44–47.

[35] S. Wilton and N. Jouppi, "Cacti: an enhanced cache access and cycle time model," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 5, pp. 677–688, 1996.

[36] H. Luo, L. Shi, Q. Li, C. J. Xue, and E. H.-M. Sha, "Energy, latency, and lifetime improvements in mlc nvm with enhanced wom code," in *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2018, pp. 554–559.

[37] Y. Zhang, L. Zhang, W. Wen, G. Sun, and Y. Chen, "Multi-level cell stt-ram: Is it realistic or just a dream?" in *2012 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2012, pp. 526–532.

[38] R. Abdel-Khalek and V. Bertacco, "Post-silicon platform for the functional diagnosis and debug of networks-on-chip," *ACM Trans. Embed. Comput. Syst.*, vol. 13, no. 3s, mar 2014. [Online]. Available: https://doi.org/10.1145/2567936

[39] ARM, "Arm cortex-a series programmer's guide for armv7-a." [Online]. Available: https://developer.arm.com/documentation/den0013/d

[40] ——, "Arm cortex-a series programmer's guide for armv7-a," march 2011. [Online]. Available: https://developer.arm.com/documentation/den0013/d

[41] ——, "Cortex-m0," january 2012. [Online]. Available: https://developer.arm.com/Processors/Cortex-M0

[42] Gaisler, "Leon3 processor," march 2018. [Online]. Available: https://developer.arm.com/Processors/Cortex-M0

[43] A. Das, A. Kumar, J. Jose, and M. Palesi, "Opportunistic caching in noc: Exploring ways to reduce miss penalty," *IEEE Transactions on Computers*, vol. 70, no. 6, pp. 892–905, 2021.

[44] N. Jindal, P. R. Panda, and S. R. Sarangi, "Reusing trace buffers to enhance cache performance," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, 2017, pp. 572–577.

[45] E. Abbasian, F. Izadinasab, and M. Gholipour, "A reliable low standby power 10t sram cell with expanded static noise margins," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 4, pp. 1606–1616, 2022.

[46] Y. Kumar, S. Sivakumar, and J. Jose, "Endura : Enhancing durability of multi level cell stt-ram based non volatile memory last level caches," in *2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC)*, 2022, pp. 1–6.

[47] R. Elnaggar, K. Basu, K. Chakrabarty, and R. Karri, "Runtime malware detection using embedded trace buffers," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 1, pp. 35–48, 2022.

[48] Y. Xie and G. H. Loh, "Scalable shared-cache management by containing thrashing workloads," in *High Performance Embedded Architectures and Compilers*, Y. N. Patt, P. Foglia, E. Duesterwald, P. Faraboschi, and X. Martorell, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 262–276.

[49] K. Raghavendra, T. S. Warrier, and M. Mutyam, "Skipcache: Miss-rate aware cache management," in *21st International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2012, pp. 481–481.

[50] S. Sivakumar, T. Abdul Khader, and J. Jose, "Improving lifetime of non-volatile memory caches by logical partitioning," in *Proceedings of the 2021 on Great Lakes Symposium on VLSI*, ser. GLSVLSI '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 123–128. [Online]. Available: https://doi.org/10.1145/3453688.3461488

List of Publications

Publications from Thesis Work:

Refereed Journals:

1. **S. Sivakumar**, John Jose, and Vijaykrishnan Narayanan, "Enhancing Lifetime and Performance of MLC NVM Caches using Embedded Trace buffers ", *ACM Transactions on Design Automation of Electronic Systems (ACM TODAES)*, Volume 29, Issue 3, Article No.: 58, pp. 1-24,, April 2024
   DOI: 10.1145/3659102

2. **S. Sivakumar** and John Jose, "Self Adaptive Logical Split Cache Techniques for Delayed Aging of NVM LLC", *ACM Transactions on Design Automation and Electronics Systems (ACM TODAES)*, Volume 28, Issue 6, Article No.: 97, pp. 1-24, October 2023.
   DOI: 10.1145/3616871

Refereed Conferences:

3. **S. Sivakumar**, Mani Mannampalli, and John Jose, "Enhancing Lifetime of Non-volatile Memory Caches by Write-Aware Techniques", *In Proceedings of 2022 on International Symposium on Devices and Circuits and Systems (ISDCS '22), Kolkata, Lecture Notes in Electrical Engineering*, vol 1004. Springer, Singapore. May 2023.
   DOI: 10.1007/978-981-99-0055-810

4. **S. Sivakumar**, T. M. Abdul Khader, and John Jose "Improving Lifetime of Non-Volatile Memory Caches by Logical Partitioning ", *In Proceedings of the 2021 on Great Lakes Symposium on VLSI (GLSVLSI '21). Association for Computing Machinery, New York, NY, USA*, 2021, pp. 123–128.
   DOI: 10.1145/3453688.3461488

Publications other than Thesis Work:

1. Yogesh Kumar, **S. Sivakumar** and John Jose "ENDURA : Enhancing Durability of Multi Level Cell STT-RAM based Non Volatile Memory Last Level Caches,", *In Proceedings of 2022 on International Conference on Very Large Scale Integration (VLSI-SoC), Patras, Greece, 2022,*, pp. 1-6, DOI: 10.1109/VLSI-SoC54400.2022.9939583.

Vitae

Sivakumar S. is a PhD scholar in the Department of Computer Science and Engineering (CSE) at the Indian Institute of Technology (IIT) Guwahati, Assam, India. He joined Vayavya Labs in Bangalore as a Senior Engineer in April 2023 and is now part of their SystemC team. Prior to joining the PhD program, he received an MTech degree in CSE from the School of Engineering, Cochin University of Science and Technology (CUSAT), Kochi, Kerala, India, in 2016, and a BTech degree in Electronics and Communication Engineering from the same institute in 2013. His team secured second place in the design contest at the VLSID conference in 2019. His primary research interests include non-volatile memory caches and in-memory computing.

Contact Information

E-mail: sivakumar@iitg.ac.in
siva17191@gmail.com

Phone: +91-8547185936

Website: https://www.iitg.ac.in/stud/sivakumar/