The Write Problem: Challenges in Implementation of STT-RAM based LLCs

A Thesis Submitted

in Partial Fulfilment of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

by

Prabuddha Sinha

(2019CSZ0008)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY ROPAR

APRIL, 2025

Dedicated to my mother Dr. Niharika Sinha, my father Sanjib Sinha, and my sister Dr. Prayukta Sinha.

Declaration of Originality

I hereby declare that the work that is being presented in the thesis entitled **The Write Problem: Challenges in Implementation of STT-RAM based LLCs** has been solely authored by me. It presents the result of my own independent investigation/research conducted during the time period from July 2019 to December 2024 under the supervision of Dr. Shirshendu Das, Assistant Professor, Indian Institute of Technology Hyderabad and Dr. Venkata Kalyan Tavva, Assistant Professor, Indian Institute of Technology Ropar. To the best of my knowledge, it is an original work, both in terms of research content and narrative, and has not been submitted or accepted elsewhere, in part or in full, for the award of any degree, diploma, fellowship, associateship, or similar title of any university or institution. Further, due credit has been attributed to the relevant state-of-the-art and collaborations (if any) with appropriate citations and acknowledgments, in line with established ethical norms and practices. I also declare that any idea/data/fact/source stated in my thesis has not been fabricated/ falsified/ misrepresented. All the principles of academic honesty and integrity have been followed. I fully understand that if the thesis is found to be unoriginal, fabricated, or plagiarized, the Institute reserves the right to withdraw the thesis from its archive and revoke the associated Degree conferred. Additionally, the Institute also reserves the right to appraise all concerned sections of society of the matter for their information and necessary action (if any). If accepted, I hereby consent for my thesis to be available online in the Institute's Open Access repository, inter-library loan, and the title & abstract to be made available to outside organizations.

Signature

Name: Prabuddha Sinha
Entry Number: 2019CSZ0008
Program: PhD
Department: Computer Science and Engineering
Indian Institute of Technology Ropar
Rupnagar, Punjab 140001

Date: April 17, 2025

Acknowledgement

First and foremost, I would like to express my deepest sense of gratitude to my doctoral advisors, Dr. Shirshendu Das, and Dr. Venkata Kalyan Tavva for their invaluable guidance, unwavering support, and constant encouragement throughout the development of this thesis. Their expertise, insights, and thoughtful feedback have been instrumental in shaping my work. I am very lucky to have them as my guides on this beautiful journey. Dr. Shirshendu's limitless patience and fatherly advice during tough times has been a beacon of hope during this journey.

I also thank my collaborators/interns Krishna Pratik BV and Mangena Likhit Sai for their constant strive for excellence. I would also like to extend my sincere thanks to the members of my doctoral committee, Dr. Neeraj Goel and Dr. Nitin Auluck, Dr. Sudeepta Mishra, and Dr. Mahendra Sakare, for their insightful comments, suggestions, and for taking the time to review my work. Your input has greatly enriched the quality of this thesis.

I am deeply thankful to my family specially my late father, mother and sister for their unconditional love and support during this challenging journey. Your patience, understanding, and encouragement have provided me with the strength to persevere.

Special thanks to my lab members and colleagues of the CA-SIG Lab at CSE Department, IIT Ropar. I would particularly thank Prathamesh Nitin Tanksale, Waqar Hassan Mir, Jaspinder Kaur, Satya Jaswant Badri, Saurabh Jaiswal, Atul Kumar, Lalit Sharma and Pravesh Jamgade for the countless discussions, moral support, and the laughter that helped me navigate through stressful times. I am also thankful to my closest friends, Napendra Solanki and Gagan Sahoo who have been a constant through all my ups and downs during this course. The time spent together helped me recharge and continue to strive for excellence.

I express my gratitude to Prof. Rajeev Ahuja, the Honourable Director, Indian Institute of Technology Ropar, and the administration for providing all necessary administrative support and technical facilities required for carrying out the present research work. I also express my deep gratitude to the Head and entire faculty and staff members of the Department of Computer Science and Engineering, Indian Institute of Technology Ropar, for their continuous support and encouragement. Lastly, I would like to acknowledge the financial support provided by Qualcomm Incorporated through Qualcomm Faculty Award 2022 to my supervisor, Dr. Venkata Kalyan Tavva, without which this thesis would not have been possible.

This thesis is a culmination of not just my efforts, but also the collective efforts of everyone who has been part of my academic journey. I am deeply grateful for all your contributions. Thank you.

Certificate

This is to certify that the thesis entitled **The Write Problem: Challenges in Implementation of STT-RAM based LLCs**, submitted by **Prabuddha Sinha (2019CSZ0008)** for the award of the degree of **Doctor of Philosophy** of Indian Institute of Technology Ropar, is a record of bonafide research work carried out under my guidance and supervision. To the best of my knowledge and belief, the work presented in this thesis is original and has not been submitted, either in part or fully, for the award of any other degree, diploma, fellowship, associate, or similar title from any university or institution.

In my opinion, the thesis has reached the standard of meeting the requirements of the regulations related to the degree.

Signature of the Supervisors:

Dr. Shirshendu Das
Department of CSE
Indian Institute of Technology Hyderabad
Hyderabad, Telengana, 502284

Dr. Venkata Kalyan Tavva
Department of CSE
Indian Institute of Technology Ropar
Rupnagar, Punjab, 140001

Date: April 17, 2025

Lay Summary

Non-volatile memory (NVM) based Last Level Caches (LLCs) are a viable alternative to the traditional Static Random Access Memory (SRAM) based LLCs. Implementing NVM based LLCs come with its own set of problems that mainly arises due to the writes. This thesis as the name suggests addresses and highlights the major issues that occur due to writes which prevent the implementation of an Spin Transfer Torque Random Access Memory (STT-RAM) based LLCs.

This thesis addresses the specific problems that arise due to the excessive uneven writes specifically lifetime degradation and performance degradation. The first work introduces two different techniques which aimed at wear leveling at the set level and way level granularity through write blocking and dynamic write bypassing. The second work proposes a decoupled cache architecture which is responsible for wear leveling at the block level granularity. The decoupled cache architecture also provides a platform for designing secured caches.

Another key part of the thesis is targeted endurance attacks that will be a security risk as any malicious attacker from a single core can target the shared LLC by focusing its writes to specific locations and drastically increasing the write count and thereby effecting the lifetime of the STT-RAM based LLC. The techniques are also effective against static and dynamic counter based wear leveling techniques.

Abstract

The design of Last Level Caches (LLCs) using Static Random Access Memory (SRAM) is increasingly being challenged by emerging memory technologies like Non-Volatile Memories (NVM). Among these, Spin-Transfer Torque RAM (STT-RAM) stands out for its higher density and lower static power consumption. However, its drawbacks—higher write latency, increased write power, and limited write endurance—pose significant challenges. The primary issue preventing widespread adoption of STT-RAM in LLCs is the low write endurance, largely caused by the uneven distribution of write operations across the cache. Existing techniques to address this focus on minimizing either inter-set (InterV) or intra-set (IntraV) write variation to prolong the lifetime of STT-RAM-based LLCs. Additionally, STT-RAM's high write latency can lead to congestion in the read-write queue of the LLC.

To address these challenges, two techniques have been proposed to enhance endurance while maintaining performance. The first, PROLONG, is a dynamic write bypassing approach that redirects write-backs from the L2 cache to an SRAM buffer or main memory. This decision is guided by two parameters: the write hotness of the cache set and the liveness score of the incoming block. The second, LiveWay, dynamically bypasses writes based on their placement in write-hot ways and their liveness scores. Both methods significantly improve wear leveling, reducing InterV and IntraV by approximately 84% and 53%, respectively, while achieving a Relative Lifetime Improvement (RLI) of up to 22×. Additionally, by alleviating write congestion in the read-write queue, these techniques minimize system impact, ensuring smoother performance.

Generic wear-leveling techniques primarily focus on extending LLC lifetime by reducing InterV and IntraV write variation, but block-level wear leveling remains rare. To address this gap, a decoupled cache architecture has been proposed. In this design, the Set-Associative SRAM tag array is separated from the Fully-Associative Data array, with the two linked via forward and backward pointers that maintain a 1:1 mapping. Two techniques are introduced within this architecture. The Primal Approach swaps writes between write-hot and write-cold blocks based on their write counts, with each block maintaining an individual write counter. The Hardware-Efficient Approach categorizes blocks into buckets using simple hashing. Writes from write-hot buckets are then redirected to write-cold buckets. These methods can achieve a RLI of up to 13.07×.

Malicious attacks in a multi-core setup require access to just one core to repeatedly target specific memory locations, leading to accelerated lifetime degradation. To expose this vulnerability in STT-RAM-based LLCs, we propose four distinct attacks: Recurring Location Attack (RLA), Recurring Toggle Attack (RTA), Random Location Attack (RnLA), and Random Toggle Attack (RnTA). These Targeted Endurance Attacks illustrate the impact of malicious benchmarks on modern counter-based wear-leveling techniques and reveal how wear leveling influences the effectiveness of such attacks.

Keywords: STT-RAM; Last Level Cache (LLC); InterV; IntraV; Wear leveling; Lifetime Improvement; Cache decoupling; Targeted Endurance Attack.

List of Publications

Journals

1. **Prabuddha Sinha**, Krishna Pratik BV, Shirshendu Das and Venkata Kalyan Tavva, "SmartDeCoup: Decoupling the STT-RAM LLC for even Write Distribution and Lifetime Improvement", in Elsevier Journal of System Architecture, February 2025. [DOI: https://doi.org/10.1016/j.sysarc.2025.103367]

2. **Prabuddha Sinha**, Mangena Likhit Sai, Shirshendu Das and Venkata Tavva Kalyan, "TENDRA: Targeted Endurance Attack in STT-RAM based LLCs", in IEEE Embedded Systems Letters. [DOI: https://doi.org/10.1109/LES.2024.3502297]

3. Sudershan Kumar, **Prabuddha Sinha**, and Shirshendu Das, "WinDRAM: Weak rows as in-DRAM cache", Concurrency and Computation: Practice and Experience 34, p.e.7350, 2022. [DOI: https://doi.org/10.1002/cpe.7350]

Conference Proceedings

1. **Prabuddha Sinha**, Krishna Pratik BV, Shirshendu Das and Venkata Kalyan Tavva, "PROLONG: Priority based Write Bypassing Technique for Longer Lifetime in STT-RAM based LLC", in 10th International Symposium on Memory Systems (MEMSYS), Washington DC, USA, 2024. [DOI: https://doi.org/10.1145/3695794.3695803]

2. **Prabuddha Sinha**, Krishna Pratik BV, Shirshendu Das and Venkata Kalyan Tavva, "Hide-N-Seek: Hiding Writes in Buffer for Lifetime Improvement in STT-RAM based LLC", at HiPC 2023 Student Research Symposium, in Proceedings of the 2023 IEEE 30th International Conference on High Performance Computing, Data and Analytics Workshop (HiPCW), Goa, India, 2023, pp-84. [DOI: https://doi.org/10.1109/HiPCW61695.2023.00021]

3. **Prabuddha Sinha**, Krishna Pratik BV, Shirshendu Das and Venkata Kalyan Tavva, "LiveWay: Dynamic Write Bypassing for Lifetime Enhancement in STT-RAM LLC", at HiPC 2024 Student Research Symposium, in Proceedings of the 2024 IEEE 31st International Conference on High Performance Computing, Data and Analytics Workshop (HiPCW), Bangalore, India, 2024. [DOI: https://doi.ieeecomputersociety.org/10.1109/HiPCW63042.2024.00035]

Book Chapter

1. Sudarshan Kumar, **Prabuddha Sinha** and Sujata Pal, "Crowd-Sourced Centralized Thermal Imaging for Isolation and Quarantine" in Computational Modeling and Data Analysis in COVID-19 Research, 2021, C. R. Panigrahi, B. Pati, M. Rath, and R. Buyya, Eds. CRC Press, ch. 8, p. 20. [DOI: https://doi.org/10.1201/9781003137481]

List of Abbreviations and Notations

Abbreviations

AES	Advanced Encryption Standard
APM	Adaptive Placement and Migration
CMOS	Complementary Metal-Oxide-Semiconductor
CPU	Central Processing Unit
DASCA	Dead Write Prediction Assisted STT-RAM Cache Architecture
DWAWR	Dynamic Way Aware Write Restriction
DWWR	Dynamic Window Write Restriction
DWM	Domain Wall Memory
E+R	Evict and Reload
E+T	Evict and Reload
ECDSA	Elliptic Curve Digital Signature Algorithm
ETB	Embedded Trace Buffer
EViSC	Enhanced Virtual Split Cache
F+F	Flush and Flush
F+R	Flush and Reload
FSDRP	Fellow Set with Dynamic Reserve Part
FSSRP	Fellow Set with Static Reserve Part
FT	Flush Threshold
GB	Giga Bytes
GOAS	Grouped On-access inter-set Swapping
HALLS	Highly Adaptable Last-Level STT-RAM Cache
HC	Hybrid Cache
HCA	Hybrid Cache Architecture
IBTC	Inclusive Bypass Tag Cache
IPC	Instructions executed Per Cycle
KB	Kilo Bytes
L1	Level 1
L2	Level 2
L3	Level 3
LHCA	Inter-cache Level Hybrid Cache Architecture
LLC	Last Level Cache
LRU	Least Recently Used
LRU-CB	Least Recently Used Cold Block
LSC	Liveness Score Counter
MB	Mega Bytes
MDB	Multiple Dirty Bits

MM	Main Memory
MPKI	Misses per Kilo Instructions
MTJ	Magnetic Tunnel Junction
NIPC	Normalized Instructions per Cycle
NP	Normal Part of Cache
NVM	Non-Volatile Memory
NUCA	Non-Uniform Cache Architecture
OAP	An obstruction-aware cache management policy for STT-RAM last-level caches
OAS	On-Access Inter-Set Swapping
OS	Operating System
PID	Process Identifier
PoLF	Probabilistic Set Line Flush
PTHCM	Prediction Table-Based Cache Line Replacement and Management Policy
P-ViSC	Protean Virtual Split Cache
P+P	Prime and Probe
PRO	PROLONG
RAM	Random Access Memory
RA	Recently Accessed
RCR	Recency-Aware Replacement
RFR	Refresh-Aware Replacement
RLA	Recurring Location Attack
RnLA	Random Location Attack
RnTA	Random Toggle Attack
RTA	Recurring Toggle Attack
RQ	Read Queue
RSA	Rivest–Shamir–Adleman
SRepl	Swap-on-Replacement
SBAC	Statistic-Based Write Bypassing Technique
SGX	Software Guard Extensions
SLAM	A hybrid last-level cache architecture
SRAM	Static Random Access Memory
SRW	Swap-on-Write
STT-RAM	Spin-Transfer Torque RAM
SW	Swap-on-Write
SWS	Swap Shift
TANC	Trace Buffer Assisted Non-Volatile Memory Cache
TSX	Hardware Transactional Memory
ViSC	Virtual Split Cache
VM	Virtual Machine

WAD	Writeback Aware Displacement
WALL-NVC	Write Aware Last Level Non-Volatile Cache
WCAB	Write Congestion Aware Bypass
WH	Write Hot
WIPE	Wearout Informed Pattern Elimination
WL	Write Latency
WQ	Write Queue

Notations

AE_{pro}	Additional Energy consumed by PROLONG
$BPTR$	Backward Pointer
E_{buff}	Energy consumed by SRAM buffer
E_{write}	Energy saved by reducing the number of writes
$FPTR$	Forward Pointer
H_n	n^{th} Write Hot Bucket
$InterV$	Inter-set Write Variation
$IntraV$	Intra-set Write Variation
LSC	Liveness Score Counter
LSn	LSC value is 'n'
M	Total number of cache ways
N	Total number of cache sets
RL	Relative Lifetime
RLI	Relative Lifetime Improvement
$WCBkt$	Write Cold Bucket
$WCTR$	Write Counter
$WHBkt$	Write Hot Bucket
WSC	Writes Set Counter
W_{avg}	Average Writes in a cache block
$w_{i,j}$	Writes to a specific block of i^{th} set and j^{th} block
W_{max}	Maximum Writes in a cache block

Contents

Declaration iv

Acknowledgement v

Certificate vi

Lay Summary vii

Abstract viii

List of Publications ix

List of Abbreviations and Notations x

List of Figures xvii

List of Tables xxiii

1 Introduction 1

1.1 Introduction . . . 1

1.1.1 STT-RAM LLCs . . . 2

1.1.2 Challenges and Solutions for implementing STT-RAM LLCs . . . 3

1.1.3 Write Variations . . . 3

1.1.4 Wear Leveling Techniques . . . 4

1.2 Motivation . . . 4

1.3 Summary and Organization of the Thesis . . . 5

2 Background and Literature Review 7

2.1 Background . . . 7

2.1.1 Cache Memories . . . 7

2.1.2 STT-RAM Cell . . . 8

2.1.3 STT-RAM based LLC . . . 9

2.2 Write Variation and Lifetime . . . 10

2.3 Techniques to Improve Lifetime in STT-RAM based LLCs . . . 11

2.3.1 Wear Leveling Techniques . . . 12

2.3.2 Write Bypassing Techniques . . . 15

2.3.3 Hybrid Caches . . . 16

2.3.4 Miscellaneous Techniques . . . 18

2.3.5 Summary of this Section . . . 20

2.4 Attacks on STT-RAM LLCs . . . 21

2.4.1 Cache Timing Channel Attacks . . . 21
2.4.1.1 Cache Side Channel Attacks (SCA) . . . 21
2.4.1.2 Cache Covert Channel Attacks (CCA) . . . 21
2.4.2 Cache Contention Attacks . . . 22
2.4.3 Cache Occupancy Attacks . . . 22
2.4.4 Cache Rowhammer based Attacks . . . 22
2.4.5 Cache Miscellaneous Attacks . . . 22
2.4.6 Summary of this Section . . . 23
2.5 Summary of this Chapter . . . 24

3 Dynamic Write Bypassing for Lifetime Improvement in STT-RAM LLCs 25
3.1 Introduction . . . 25
3.2 Motivation . . . 26
3.3 PartA (To reduce InterV): PROLONG: Priority based Write Bypassing Technique for Longer Lifetime in STT-RAM based LLC . . . 27
3.3.1 Proposed Architecture . . . 27
3.3.1.1 Writes set counter (WSC) . . . 28
3.3.1.2 SRAM Buffer . . . 29
3.3.1.3 Liveness Score Counter (LSC) . . . 29
3.3.2 Working of PROLONG . . . 30
3.3.3 PROLONG Algorithm . . . 33
3.4 PartB (To reduce IntraV): LiveWay: Dynamic Write Bypassing for Lifetime Enhancement in STT-RAM LLC . . . 33
3.4.1 Proposed Architecture . . . 33
3.4.2 Working of LiveWay . . . 34
3.5 Experimental Setup . . . 35
3.5.1 Simulator Setup . . . 35
3.5.2 Workloads . . . 36
3.6 PartA: Results and Analysis . . . 37
3.6.1 Single-core Analysis . . . 37
3.6.2 Multi-core Analysis . . . 41
3.6.3 Sensitivity Analysis . . . 42
3.6.3.1 Importance of SRAM buffer . . . 44
3.6.3.2 Comparison with other write bypassing techniques . . . 44
3.6.3.3 Lifetime Comparison Analysis . . . 46
3.6.3.4 SPEC2017 vs SPEC2006 vs GAP . . . 47
3.6.4 Hardware Overhead and Energy Consumption . . . 47
3.7 PartB: Results and Analysis . . . 49
3.7.1 Single-core Analysis . . . 49
3.7.2 Multi-core Analysis . . . 50
3.7.3 SPEC2017 vs SPEC2006 vs GAP . . . 51

3.7.4 Hardware Overhead and Energy Consumption . . . 52
3.8 Conclusion . . . 54

4 Decoupling the tag and data array for Lifetime Improvement 55
4.1 Introduction . . . 55
4.2 Motivation . . . 55
4.3 SmartDeCoup . . . 57
4.3.1 LLC Organization . . . 57
4.3.1.1 Maintaining Coherence: . . . 58
4.3.2 Primal Approach . . . 59
4.3.2.1 Working Example . . . 60
4.3.2.2 Drawbacks of Primal Approach . . . 61
4.3.3 Hardware Efficient Approach . . . 61
4.3.3.1 Bucket formation . . . 62
4.3.3.2 Working Example . . . 62
4.3.4 Primal Approach vs Hardware Efficient Approach . . . 64
4.3.5 Importance of SmartDeCoup in Modern LLC . . . 64
4.3.5.1 Challenges in Implementing SmartDeCoup on Mirage and Maya-Cache: . . . 65
4.3.5.2 SmartDeCoup with non-decoupled secured LLC designs . . . 65
4.4 Experiments . . . 66
4.4.1 Simulator Setup . . . 66
4.4.2 Workloads . . . 68
4.5 Results and Analysis . . . 68
4.5.1 Primal Approach . . . 69
4.5.1.1 Single-core Analysis . . . 70
4.5.1.2 Multi-core Analysis . . . 72
4.5.2 Hardware Efficient Approach . . . 74
4.5.2.1 Single-core Analysis . . . 74
4.5.2.2 Multi-core Analysis . . . 76
4.5.3 Sensitivity Analysis . . . 78
4.5.3.1 Primal Approach vs Hardware Efficient Approach . . . 78
4.5.3.2 SPEC2017 vs SPEC2006 vs GAP . . . 80
4.5.3.3 Analysis of AI Workloads . . . 82
4.5.4 Overhead Analysis . . . 83
4.6 Conclusion . . . 85

5 Endurance Attacks on STTRAM LLCs 87
5.1 Introduction . . . 87
5.2 Motivation . . . 88
5.3 TENDRA: Targeted Endurance Attack . . . 89
5.3.1 Threat Model . . . 89

5.3.2 The Attack Idea . . . 90
5.3.2.1 Recurring Location Attack (RLA) . . . 90
5.3.2.2 Recurring Toggle Attack (RTA) . . . 91
5.3.2.3 Random Location Attack (RnLA) . . . 92
5.3.2.4 Random Toggle Attack (RnTA) . . . 93
5.3.3 Comparison between Attacks . . . 94
5.4 Experimental Evaluation . . . 96
5.4.1 Simulator Setup . . . 96
5.4.2 Workloads . . . 96
5.5 Results and Analysis . . . 97
5.5.1 Effects of RLA, RTA, RnLA and RnTA on different State-of-the-art Wear Leveling Techniques . . . 97
5.5.1.1 Write Count . . . 107
5.5.2 Effects on Lifetime and Performance . . . 108
5.6 Conclusion and Future Work . . . 109

6 Conclusion and Future Scope **111**
6.1 Major Research Contributions . . . 111
6.2 Future Scope . . . 112
6.2.1 Practical Lifetime Improvement for Industry-wide Adaptation . . . 113

References **115**

List of Figures

1.1 Memory Organization and Hierarchy. 1

2.1 A generic Set-Associative LLC. 8
2.2 Layout of an STT-RAM cell. 9

3.1 Reduction in InterV and IntraV for various state-of-the-art wear leveling techniques. 26
3.2 IPC Degradation for various state-of-the-art wear leveling techniques. . . . 26
3.3 Non-uniform write distribution across cache sets 28
3.4 Non-uniform write distribution across cache ways 28
3.5 Average Write Distribution in STT-RAM based LLC with LRU among the Write Hot (WH) buckets (15%) with Liveness Score (LS). 32
3.6 Architecture and Steps in PROLONG Algorithm. 33
3.7 Proposed Architecture of LiveWay. 34
3.8 Flowchart of the Proposed Algorithm for LiveWay. 34
3.9 Percentage Reduction in InterV in Single-core System (higher the better). . 37
3.10 Percentage Reduction in IntraV in Single-core System (higher the better). . 38
3.11 Comparison of Normalized Total Write Count in single-core Systems (lower is better). 39
3.12 Comparison of Relative Lifetime Improvement in single-core Systems (higher the better). 39
3.13 Comparison of Normalized IPC in single-core Systems (higher the better). . 40
3.14 Average Comparison of PROLONG in Single-core Systems with state-of-the-art. 40
3.15 Percentage Reduction in InterV for Multicore Systems (higher the better). . 41
3.16 Percentage Reduction in IntraV for Multi-core Systems (higher the better). 41
3.17 Relative Lifetime Improvement (RLI) for Multi-core Systems (higher the better). 41
3.18 Normalized Total Writes w.r.t. baseline (LRU) for Multi-core Systems (lower is better). 42
3.19 Normalized IPC w.r.t. baseline (LRU) for Multi-core Systems (higher the better). 42
3.20 Comparison of different PROLONG configurations with baseline in a single-core environment. PRO(x%) means the bypass aggressiveness of PROLONG is x%. Each bypass aggressiveness value is experimented with 32KB, 64KB, 128KB, 256KB, and 512KB of buffer size. 43

3.21 Buffer hit per write bypass in different configurations of PROLONG (higher the better). . . . 44
3.22 Buffer hit per write bypass in different configurations of PROLONG for AI workloads (higher the better). . . . 45
3.23 Comparison of Normalized Lifetime of PROLONG with Density (higher the better). . . . 45
3.24 Comparison of Normalized IPC of PROLONG with Density (higher the better). . . . 45
3.25 Normalized IPC of various different benchmarks run on various configurations. 48
3.26 Reduction in InterV of various different benchmarks run on various configurations. . . . 48
3.27 Reduction in IntraV of various different benchmarks run on various configurations. . . . 48
3.28 RLI of various different benchmarks run on various configurations. . . . 49
3.29 % Reduction in IntraV for Single-core Systems (Higher the better). . . . 49
3.30 Relative Lifetime Improvement (RLI) (in times) for Single-core Systems (Higher the better). . . . 50
3.31 Geomean Normalized IPC for Single-core Systems (Higher the better). . . . 50
3.32 % Reduction in IntraV for Multi-core Systems. . . . 51
3.33 Relative LI (in times) for Multi-core Systems. . . . 51
3.34 Normalized IPC for Multi-core Systems. . . . 52
3.35 Normalized IPC of various different benchmarks run on various configurations. 53
3.36 Reduction in IntraV of various different benchmarks run on various configurations. . . . 53
3.37 RLI of various different benchmarks run on various configurations. . . . 53

4.1 Heatmap representing the non-uniform write distribution at LLC for a single-core *milc* run. . . . 56
4.2 The proposed decoupled cache. . . . 59
4.3 Working Example of the Primal Approach. . . . 61
4.4 Working Example of the Hardware Efficient Approach. . . . 63
4.5 Write Distribution for Epoch 1 for *Primal Approach on SWAP 10%*. . . . 69
4.6 Write Distribution for Epoch 2 for *Primal Approach on SWAP 10%*. . . . 69
4.7 Write Distribution for Epoch 3 for *Primal Approach on SWAP 10%*. . . . 69
4.8 Relative Lifetime Improvement in single-core systems for *Primal Approach*. 70
4.9 Percentage reduction of IntraV in single-core systems for *Primal Approach*. 71
4.10 Percentage reduction of InterV in single-core systems for *Primal Approach*. 71
4.11 Normalized IPC in single-core systems for *Primal Approach*. . . . 71
4.12 Relative Lifetime Improvement in multi-core systems for *Primal Approach*. 72
4.13 Percentage reduction of IntraV in multi-core systems for *Primal Approach*. 72
4.14 Percentage reduction of InterV in multi-core systems for *Primal Approach*. . 73
4.15 Normalized IPC in multi-core systems for *Primal Approach*. . . . 73

4.16 Relative Lifetime Improvement in single-core systems for *Hardware Efficient Approach*. . . . 74
4.17 Percentage Reduction of IntraV in single-core systems for *Hardware Efficient Approach*. . . . 75
4.18 Percentage Reduction of InterV in single-core systems for *Hardware Efficient Approach*. . . . 75
4.19 Normalized IPC in single-core systems for *Hardware Efficient Approach*. . . 75
4.20 Relative Lifetime Improvement in multi-core systems for *Hardware Efficient Approach*. . . . 77
4.21 Percentage Reduction of IntraV in multi-core systems for *Hardware Efficient Approach*. . . . 77
4.22 Percentage Reduction of InterV in multi-core systems for *Hardware Efficient Approach*. . . . 77
4.23 Normalized IPC in multi-core systems for *Hardware Efficient Approach*. . . 77
4.24 Comparison of Reduction in IntraV and InterV in a Single-core System. . . 79
4.25 Comparison of Redirection/SWAP Count in a Single-core System. 79
4.26 Comparison of Relative Lifetime Improvement in a Single-core System. . . . 79
4.27 Comparison of Normalized IPC in a Single-core System. 79
4.28 Normalized IPC of various different benchmarks run on various configurations of our proposed approaches. . . . 81
4.29 Reduction in IntraV of various different benchmarks run on various configurations of our proposed approaches. . . . 82
4.30 Reduction in InterV of various different benchmarks run on various configurations of our proposed approaches. . . . 82
4.31 RLI of various different benchmarks run on various configurations of our proposed approaches. . . . 82
4.32 RLI of AI workloads on various configurations. . . . 83
4.33 % Reduction of InterV and IntraV of AI workloads on various configurations. 83

5.1 Visualization of the Different Types of Targeted Endurance Attacks. 95
5.2 LRU-Mix0. . . . 98
5.3 i^{2}WAP-Mix0. . . . 98
5.4 SWWR-Mix0. . . . 98
5.5 DWAWR-Mix0. . . . 98
5.6 PROLONG-Mix0. . . . 98
5.7 LiveWay-Mix0. . . . 98
5.8 SDC:Primal-Mix0. . . . 98
5.9 SDC:HE-Mix0 98
5.10 LRU-Mix1. . . . 99
5.11 i^{2}WAP-Mix1. . . . 99
5.12 SWWR-Mix1. . . . 99
5.13 DWAWR-Mix1. . . . 99

5.14 PROLONG-Mix1. 99
5.15 LiveWay-Mix1. 99
5.16 SDC:Primal-Mix1. 99
5.17 SDC:HE-Mix1. 99
5.18 LRU-Mix2. 100
5.19 i^{2}WAP-Mix2. 100
5.20 SWWR-Mix2. 100
5.21 DWAWR-Mix2. 100
5.22 PROLONG-Mix2. 100
5.23 LiveWay-Mix2. 100
5.24 SDC:Primal-Mix2. 100
5.25 SDC:HE-Mix2. 100
5.26 LRU-Mix3. 101
5.27 i^{2}WAP-Mix3. 101
5.28 SWWR-Mix3. 101
5.29 DWAWR-Mix3. 101
5.30 PROLONG-Mix3. 101
5.31 LiveWay-Mix3. 101
5.32 SDC:Primal-Mix3. 101
5.33 SDC:HE-Mix3. 101
5.34 LRU-Mix4. 102
5.35 i^{2}WAP-Mix4. 102
5.36 SWWR-Mix4. 102
5.37 DWAWR-Mix4. 102
5.38 PROLONG-Mix4. 102
5.39 LiveWay-Mix4. 102
5.40 SDC:Primal-Mix4. 102
5.41 SDC:HE-Mix4. 102
5.42 LRU-Mix5. 103
5.43 i^{2}WAP-Mix5. 103
5.44 SWWR-Mix5. 103
5.45 DWAWR-Mix5. 103
5.46 PROLONG-Mix5. 103
5.47 LiveWay-Mix5. 103
5.48 SDC:Primal-Mix5. 103
5.49 SDC:HE-Mix5. 103
5.50 LRU-Mix6. 104
5.51 i^{2}WAP-Mix6. 104
5.52 SWWR-Mix6. 104
5.53 DWAWR-Mix6. 104
5.54 PROLONG-Mix6. 104

5.55 LiveWay-Mix6. 104
5.56 SDC:Primal-Mix6. 104
5.57 SDC:HE-Mix6. 104
5.58 LRU-Mix7. 105
5.59 i^{2}WAP-Mix7. 105
5.60 SWWR-Mix7. 105
5.61 DWAWR-Mix7. 105
5.62 PROLONG-Mix7. 105
5.63 LiveWay-Mix7. 105
5.64 SDC:Primal-Mix7. 105
5.65 SDC:HE-Mix7. 105
5.66 Degradation in Lifetime. 108
5.67 Degradation in IPC. 109

6.1 Graphical Overview of Thesis 111

List of Tables

2.1 Comparison of SRAM cell and STT-RAM cell (R/W=Read/Write). 10
2.2 Classification and Brief Overview of Lifetime Enhancement Techniques for NVM based LLC (LI=Lifetime Improvement, RedInterV=Reduction in InterV, RedIntraV=Reduction in IntraV and PI=Performance Improvement and Energy Reduction=ER). 20
2.3 Classification of LLC Attacks. 23

3.1 Bucket Distribution among sets based on Liveness Score (LS). 30
3.2 Bucket Distribution among sets based on Write Counter. 31
3.3 Write Bypassing based on Write Hot Sets (WH) and LSC Buckets. 32
3.4 Simulation parameters of the baseline single-core system. 35
3.5 Different PROLONG configurations used. 36
3.6 Workloads for single-core system. 36
3.7 Workloads for Quad-Core System. 37
3.8 The important hardware parameters of the SRAM buffer used in PROLONG. The parameters are extracted from CACTI [1]. 43
3.9 Normalized Lifetime of Write Intensive Workloads. 46
3.10 Storage Overhead of Prolong. 49
3.11 Storage Overhead of LiveWay. 54

4.1 Simulation parameters of the baseline single-core system. 66
4.2 Configurations of *Primal* and *Hardware Efficient Approaches*. 67
4.3 Thresholds for Switching in different bucket count for *Hardware Efficient Approaches*. 67
4.4 Write Intensive Workloads for single-core system. 68
4.5 Workloads for Quad-Core System. 68
4.6 Geomean Comparisons for Single-core Systems with *Primal Approach*. . . . 72
4.7 Geomean Comparisons for Multi-core Systems with *Primal Approach*. . . . 74
4.8 Geomean Comparisons for Single-core Systems with *Hardware Efficient Approach*. 76
4.9 Geomean Comparisons for Multi-core Systems with *Hardware Efficient Approach*. 78
4.10 Comparisons of various configurations of the proposed technique with varying cache sizes. 80
4.11 Storage Overhead of Proposed Techniques. 84

5.1 Different TENDRA Comparisons. 94
5.2 Simulation parameters of the baseline multi-core system. 96

5.3 Workloads for Quad-Core System. 97
5.4 The maximum write count of a LLC block. 107

Chapter 1

Introduction

This chapter presents a broad overview of the problems plaguing Spin Transfer Torque Random Access Memory based Last Level Caches. This chapter delves into the significance of the proposed works, offering a brief yet insightful summary of the principal ideas and contributions.

1.1 Introduction

The advent of Machine Learning and Artificial Intelligence introduces data-intensive workloads that demand fast training and testing time. These workloads are distributed across multiple cores within the processor, all of which are integrated onto a single chip [2]. Multi-core architectures usually comprise a hierarchy of cache memories. The overall system's speed is highly dependent on the performance and capabilities of the caches as it provides a bridge to fill the gap in the performance difference between the slow Maim Memory and the fast processor. Cache memories are fast memories that can be placed on-chip. These memory systems operate on the principle of locality, storing data that is likely to be accessed in the near future. Modern commercial processors typically feature multiple levels of cache. The higher-level caches: L1 and L2 are exclusive to individual cores, while the Last Level Cache (LLC) is shared across all processing cores. The Level 1 (L1) cache is divided into two parts: the data cache (L1-D) and the instruction cache (L1-I). Figure 1.1 gives us an overview of the memory organization and hierarchy.

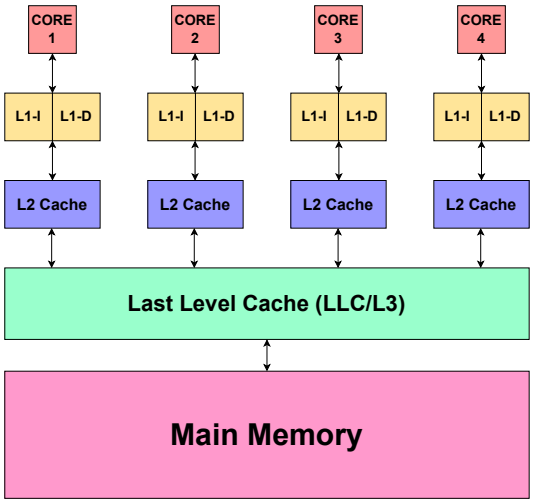

Figure 1.1: Memory Organization and Hierarchy.

Memory system design is increasingly scrutinized as architects tackle rising power demands. The growing costs of acquiring and operating large-scale supercomputers and data centers have highlighted the significant power consumption of memory subsystems, making them a central focus for efficiency improvements [3]. Supporting such workloads requires larger on-chip caches to minimize the off-chip data movement that incurs extra latency. CPU architectures have been revolutionized over the years, but the traditional cache architectures are not able to scale up as such, creating a performance bottleneck. Therefore, data-intensive workloads require larger memory systems and cost-effective alternative memory technologies that can help scale these memory systems. Traditionally, Static Random Access Memory (SRAM) has been used to design the on-chip caches. Major drawbacks of SRAM include a larger on-chip area and high leakage power. Previous studies [4] have indicated that leakage accounts for nearly 80% of the LLCs power consumption. Therefore, increasing the size of the SRAM based LLCs is not feasible with regard to power, area, and cost. Emerging memory technologies, such as embedded dynamic random access memory (eDRAM) [5], spin transfer torque random access memory (STT-RAM) [6], resistive random access memory (RRAM) [7], phase change memory (PCM) [8], and domain wall memory (DWM) [9], offer promising solutions to address these challenges due to their desirable features. When employed in the design of on-chip caches, these technologies offer significantly higher density and lower leakage compared to SRAM. This allows for reduced area and power consumption at equivalent capacities, or increased cache capacity within the same physical footprint [10].

1.1.1 STT-RAM LLCs

STT-RAM based LLC can be considered as the most expected possible alternative to SRAM based LLC because of its compatibility with the processor's fabrication technology as it is made of similar materials. More details are given in Section 2. Although there are a lot of advantages that support the use of STT-RAM LLCs the major concern about it being accepted industry-wide is the limited write endurance of an STT-RAM cell [11]. This major concern is further heightened by non-uniform write patterns across the whole LLC in multi-core architectures. These non-uniform write patterns give rise to the write variations (WVs) in the LLC [12]. Due to the fluctuations in write patterns, applications create write hot spots within the cache. This causes some locations in the STT-RAM LLC to be written into more than normal. Therefore, those locations will wear out faster, resulting in a reduction in the lifetime of the STT-RAM LLCs. Uneven write variation can also be exploited in non-secure LLCs by attackers to target the LLC through an endurance attack to reduce its lifetime. Furthermore, write operations are problematic as they take a long time to be completed as compared to a read operation, therefore it creates a congestion in the read/write queue of the LLC because when the write operation is being serviced the read operations have to wait. Therefore, it results in a drop in the performance of the system [13].

1.1.2 Challenges and Solutions for implementing STT-RAM LLCs

The major challenge in implementing the STT-RAM as LLC is the write operation and the problems associated with it [14]. These include:

1. High Write Latency.
2. High Write Energy Consumption.
3. Low Write Endurance of an STT-RAM cell.

In order to deal with the problems caused by writes to the STT-RAM LLC, researchers have identified four principal types of solutions:

- Write Variation leveling (wear leveling) - Enhances Endurance: These techniques aim to redistribute the writes evenly across the whole cache thereby reducing the maximum write count in a cache line [12, 15, 16, 14, 17, 18].
- Write Bypassing (write reduction) - Reduces Write Latency: These techniques try to bypass writes that are not useful directly to the main memory such that it leads to decongestion in the read/write queue of the LLC [19, 20, 21, 13, 22].
- Hybrid Caches (write redirection) - Enhances Endurance: It consists of two parts: a volatile section and a non-volatile section, and the majority of the techniques try to redirect the writes in the non-volatile section to the volatile section [23, 24, 25, 26, 27].
- Multi-retention time STT-RAM - Reduce Write Energy Consumption: Here different types of STT-RAM cells are used with multiple retention times and thereby varying endurance. Lower retention time cells have higher endurance and low write energy consumption [28, 29, 30].

More details about these solutions and the different state-of-the-art approaches proposed are discussed in Chapter 2.

1.1.3 Write Variations

Write Variation in an STT-RAM based set-associative LLC is generally categorized as [12]:

- *Inter-set Write Variation (InterV)*: InterV occurs exclusively within a cache set, arising from disproportionate write frequencies among the individual blocks contained within the specific cache set. Notably, certain blocks within the set undergo a higher volume of writes than others, resulting in accelerated wear for these heavily utilized blocks as compared to their counterparts.
- *Intra-set Write Variation (IntraV)*: Conversely, IntraV denotes disparate write counts observed among cache sets, highlighting a lack of uniformity in data writing patterns across the cache sets. Consequently, this imbalance accelerates the degradation of heavily utilized sets, leading to their premature breakdown relative to their less active counterparts.

1.1.4 Wear Leveling Techniques

Over the years, a multitude of wear-leveling techniques have been proposed to reduce both InterV and IntraV, with the latter comprising the majority of research. Wang et al. [12] introduced Probabilistic Set Line Flush (PoLF), which implements write blocking to a designated block within the set. It designates a block as invalid following a predefined number of writes, known as the Flush Threshold (FT). Should the number of writes to the block exceed the FT, it is blocked for further write operations. Jokar et al. [15] introduce Write-back aware intra-set displacement (WAD), an approach for identifying and managing blocks within a set that experiences limited write traffic from the main memory, utilizing a sophisticated counter mechanism. The write-intensive blocks are subsequently exchanged with a victim block within the identical set, including a process to mitigate any substantial increase in the LLC miss ratio. EqualChance, outlined by Mittal et al. [16], adopts a method that involves the strategic swapping of write-intensive blocks within the cache set with those that are either invalid or clean. Agarwal et al. [14] propose three distinct methodologies: Static Window Write Restriction (SWWR), Dynamic Window Write Restriction (DWWR), and Dynamic Way Aware Write Restriction (DWAWR). The majority of these techniques are counter based techniques and are vulnerable to intelligent endurance attacks to some extent.

The main goal of this thesis is to propose efficient wear leveling techniques that will enhance the endurance of the STT-RAM LLC while reducing the write variations and not compromising too much on the performance. We also try to exploit the vulnerability of the wear leveling techniques with the help of unique targeted endurance attacks.

1.2 Motivation

All the wear leveling techniques have always been implemented to reduce the InterV and IntraV. They help in improving the lifetime of the system, but they always have a drawback of extra performance overhead that arises due to write redirection from the write hot cache lines to another cache line. The degradation in performance is a quite significant issue and hinders the efficiency of the system. Furthermore, there are write-bypassing techniques that focus on bypassing write from the read/write queue of the LLC. This will help in improving the congestion of the read/write queue which ultimately helps in improving the performance of the system. Therefore, in order to incorporate the benefits of enhanced endurance with performance improvement we have implemented techniques that do both write bypassing and wear leveling. As per our knowledge, this is the first work to pinpoint how the drawbacks of wear leveling can be effectively resolved by efficient priority-based intelligent write bypassing. Furthermore, the majority of the wear leveling techniques mainly focus on the set-level granularity wear leveling, i.e. reduction in InterV or way-level

granularity wear leveling, i.e. reduction in IntraV, but none of the works have targeted the block-level granularity wear leveling. Our work helps in implementing block-level wear leveling in an Optimal and Hardware Efficient manner. Furthermore, these counter based wear leveling techniques may be able to wear level writes in generic workloads, but they may not be effective against targeted endurance attacks that are intelligent with write patterns that can counteract the wear leveling techniques.

1.3 Summary and Organization of the Thesis

As previously stated above the main motivation of this thesis is to identify potential problems in implementing STT-RAM as LLCs and providing mitigation techniques in order to overcome these problems. Various state-of-the-art wear leveling techniques have been proposed over the years that mainly wear level the writes and improve the lifetime by reducing the InterV and IntraV of the LLC. The redirection caused due to wear leveling brings about some lifetime degradation. However, in this work we propose a technique that aims to dynamically reduce the write count and perform wear leveling. Furthermore, we also propose a wear leveling technique for block-level granularity that tries to perfectly even out the writes. A few vulnerabilities of implementing STT-RAM with and without wear leveling techniques have been exposed by proposing some innovative endurance attacks at the STT-RAM based LLC. The work in this thesis can be divided into three major parts:

A) **Literature Survey** - In this section, we present a comprehensive survey of various cache endurance challenges, which we categorize into four key areas: write variation leveling techniques (wear leveling), write bypassing techniques (write reduction), hybrid cache architectures, and LLC attacks.

B) **Lifetime Improvement in STT-RAM LLCs** - In this section, we have analyzed the effects of uneven write distribution on lifetime of STT-RAM based LLCs. Therefore, we have explored the possibility of wear leveling without redirection and just bypassing from write hot cache locations. The writes are bypassed either to a small SRAM buffer or to the Main Memory. The small SRAM buffer is used to store those writes that have high liveness score, this will help in reducing the miss penalty that will occur if we have to fetch a block from the Main Memory. This was done at the set level and way level granularity.

   Furthermore, we also analyze the need for wear leveling at the block level granularity which is an improvement over set level wear leveling or way level wear leveling. It also looks into ways how to wear level the cache while preventing any timing channel attacks at LLC level. We propose two different techniques, one which employs a block wise write counter and wear levels the writes evenly, the other is a hardware efficient approach which clubs blocks into buckets and the writes are distributed among these buckets based on their write count.

C) **Endurance Attacks on STT-RAM LLCs** - This section shows the vulnerability of the STT-RAM LLCs against targeted endurance attacks even when implemented with wear leveling techniques.

Overall, the ideas proposed in this thesis provide efficient wear leveling techniques for lifetime improvement and also exploit the vulnerability of the STT-RAM LLC to introduce targeted endurance attacks. In order to keep this chapter short and simple, we have omitted a detailed discussion of these ideas. The organization of the thesis is outlined as follows:

- **Chapter 1: Introduction** - This chapter provides a concise introduction to multi-core caches and explores the potential implementation of STT-RAM in LLCs. It also outlines the motivation behind this thesis and its overall workflow.
- **Chapter 2: Background and Literature Review** - This chapter covers essential background information and includes a comprehensive literature survey (Part A).
- **Chapter 3: Dynamic Write Bypassing for Lifetime Improvement in STT-RAM LLCs** - This chapters discusses the ideas proposed as Part B above.
- **Chapter 4: Decoupling the tag and data array for Lifetime Improvement** - This chapter explains the ideas of Part B as mentioned above.
- **Chapter 5: Endurance Attacks on STTRAM LLCs** - This chapter discusses the idea of above mentioned Part C.
- **Chapter 6: Conclusion and Future Work** - This thesis is concluded in this chapter.

Chapter 2

Background and Literature Review

In this chapter, we have explored essential background information relevant to this thesis. It is also accompanied by a comprehensive literature review on non-volatile memory-based cache endurance. The chapter concludes with a summary of the limitations in current techniques and highlights the significance of the contributions made in this thesis.

2.1 Background

This section highlights the characteristics of STT-RAM, the NVM that we have used to design the LLC. It further discusses the challenges in implementing STT-RAM LLCs and the architectural changes required to build the same. Some generic studies in NVM like [31, 32, 33] have pointed out the challenging areas that need to be addressed.

2.1.1 Cache Memories

Modern processors are designed with multiple levels of cache to enhance memory access speed and minimize latency. In most contemporary processors, each core is equipped with private upper-level caches, while the LLC is shared among all cores. The caches considered in this work are set-associative, meaning each cache contains a fixed number of sets. A block is mapped to a specific set based on the set-index bits derived from the block's address, and the number of blocks that can reside in a set is determined by the cache's associativity. In an N-way set-associative cache, each set can hold N blocks. Throughout this thesis, N represents the cache's associativity, and M denotes the number of sets in the LLC. Figure 2.1 depicts the structure of the set-associative cache that is used as the LLC.

To access data in the cache, a memory address is divided into three components: the *tag, block offset*, and *set index*. The set index indicates the set where the block is located, the tag uniquely identifies the block within that set, and the offset specifies the location of the data within the block. It's important to note that while the L1 cache is typically virtually indexed—using the virtual address to determine the set index—the LLC is usually physically indexed, meaning the set index is derived from the physical address.

A cache replacement policy is employed to manage the selection of cache entries for eviction when a new block needs to be inserted into a non-empty set. Such a policy consists of three key components: *insertion*, *promotion*, and *eviction*. The insertion component determines where a newly arrived block will be placed in the cache. The promotion component activates upon a cache hit, boosting the priority or position of the accessed block. The

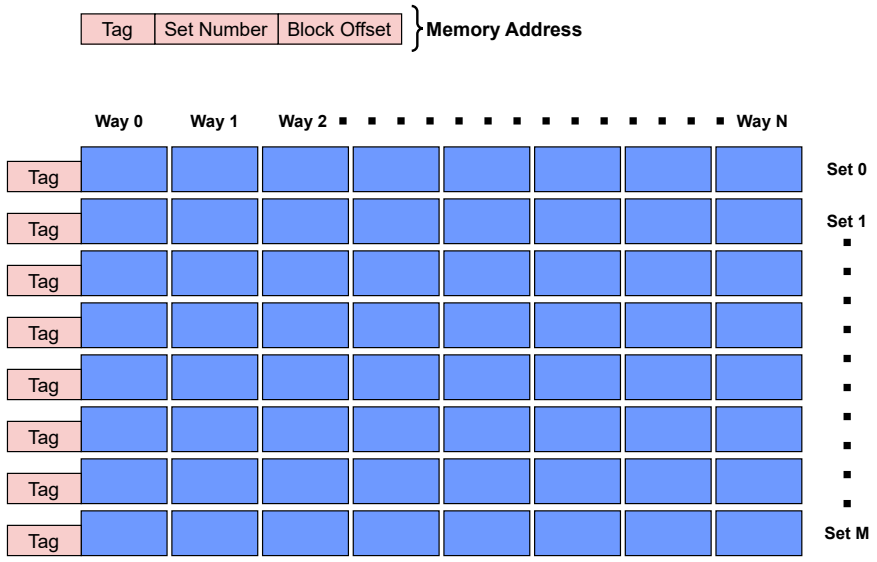

Figure 2.1: A generic Set-Associative LLC.

eviction component is responsible for selecting a block, known as the *victim block*, for replacement when necessary. In standard replacement policies, any process can evict another process's block if it is chosen as the victim. However, when processes are not permitted to evict each other's blocks, they are considered to be isolated within the cache. A LLC can be designed as inclusive, exclusive, or non-inclusive. In an inclusive LLC, the cache contains all the blocks stored in the upper-level private caches, meaning any block evicted from the LLC must also be removed from the private caches to maintain this inclusivity. In contrast, an exclusive LLC does not duplicate blocks held in the private caches, with directories used within the LLC to ensure cache coherence and track which blocks are present in the private caches. A non-inclusive cache configuration does not strictly enforce inclusion, allowing for more flexibility—blocks may reside in both the LLC and private caches, but this is not guaranteed.
Understanding these cache architectures is essential for analyzing the problem of endurance of an STT-RAM LLC and the mitigation strategies and attack directions that have been proposed.

2.1.2 STT-RAM Cell

Figure 2.2 (a) shows the conceptual layout of an STT-RAM cell. This cell comprises of an access transistor alongside a Magnetic Tunnel Junction (MTJ). The MTJ structure consists of a tunnel barrier, composed of MgO, sandwiched between two ferromagnetic layers. The reference layer, possesses a fixed magnetization direction. Conversely, the magnetization direction of the other layer, termed the free layer, is variable and can be altered by a spin-polarized current. The tunnel barrier serves as a thin insulating layer separating the two ferromagnetic layers. These layers magnetization directions encode the stored data bit within the cell. Specifically, when the magnetization directions are

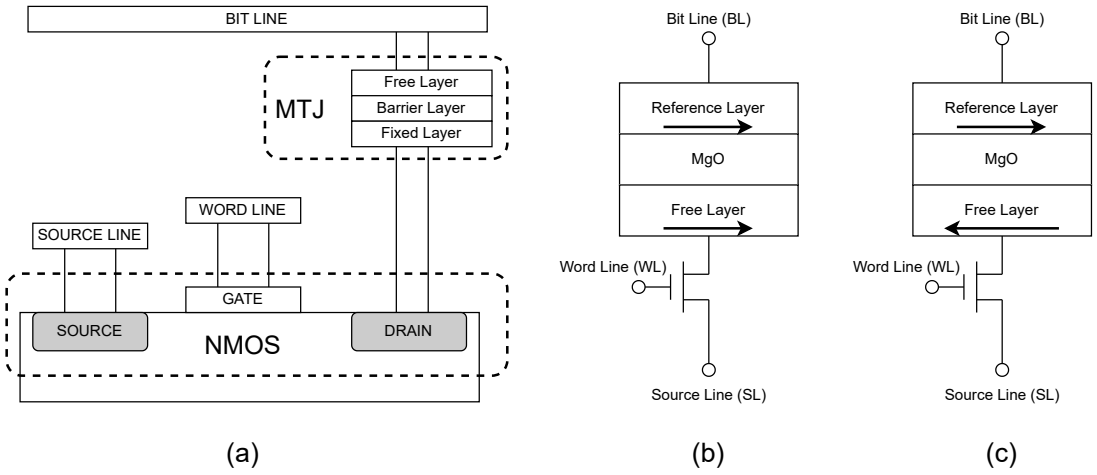

Figure 2.2: Layout of an STT-RAM cell.

anti-parallel, the resulting high resistance represents a logical '1' as shown in Figure 2.2 (c), whereas parallel magnetization denotes a logical '0' with low resistance as shown in Figure 2.2 (b).

Both read and write operations in an STT (Spin-Transfer Torque) memory cell are executed by manipulating the voltage differential between the source and a bit-line. Writing a '0' in the STT cell requires applying a significant positive voltage across the source and the bit-line, which induces a current that alters the magnetic orientation of the cell, setting it to the desired state. To write a '1', a substantial negative voltage is applied, reversing the magnetic polarity to reflect the '1' state.

In contrast, the read operation involves applying a smaller voltage across the source and the bit-line, generating a current that flows through the cell. This current is then compared against a reference current, allowing the system to determine whether the cell holds a '0' or '1'. By carefully measuring this current relative to the reference, the memory state is accurately identified without disrupting the stored data.

2.1.3 STT-RAM based LLC

STT-RAM provides numerous advantages compared to traditional SRAM-based LLCs [10]. Table 2.1 highlights the key characteristics of an STT-RAM cell in comparison to an SRAM cell. The parameter F in the table denotes the smallest feature size achievable within a specific technology node, offering a direct comparison of the cells' properties under equivalent conditions. This comparison provides insight into the differences in performance, scalability, and efficiency between STT-RAM and SRAM technologies.

Although STT-RAM boasts numerous advantages over SRAM, including smaller cell size, reduced leakage power, and extended retention time, as illustrated in Table 2.1, it has a few drawbacks. These benefits must be weighed against certain inherent disadvantages accompanying STT-RAM technology, which may impact its overall suitability. The primary challenge for on-chip memory systems that rely on STT-RAM based LLCs is the

Table 2.1: Comparison of SRAM cell and STT-RAM cell (R/W=Read/Write).

Parameters	**SRAM**	**STT-RAM**
Cell Size(F^2)	120-200	6-50
Write Endurance	10^{16}	4×10^{12}
Speed(Read/Write)	Very Fast	Very Fast/Slow
Leakage Power	High	Low
Dynamic Energy(R/W)	Low	Low/High
Retention Period	Very Low	High

significantly higher write latency and energy consumption associated with write operations compared to the traditional SRAM. While STT-RAM offers several advantages, these elevated write metrics can hinder its overall performance and efficiency, making it a critical factor to consider in the design of memory architectures. This trade-off between the benefits of STT-RAM and its inherent write-related drawbacks is a key consideration in determining its viability for widespread implementation.

2.2 Write Variation and Lifetime

Write variation (WV) presents a substantial challenge in the design of cache and non-volatile memory technologies. When write operations are unevenly distributed across the memory cells, certain areas experience a much higher frequency of writes than others. This imbalance accelerates wear and tear on those heavily written areas, leading to premature degradation of the memory cells. As a result, the overall lifespan of the cache is significantly diminished, compromising the reliability and effectiveness of the memory system. Addressing WV is crucial for ensuring that these technologies can maintain their performance and longevity in real-world applications. Even a small fraction of heavily written memory cells can render an entire cache or memory system inoperative, despite the majority of cells being far from wear-out. Our work focuses on minimizing the WV in the STT-RAM LLC. WV in an NVM-based set-associative LLC is generally categorized as:

a) *Inter-set Write Variation* ***(InterV)***

b) *Intra-set Write Variation* ***(IntraV)***

IntraV occurs exclusively within a cache set, arising from disproportionate write frequencies among the individual blocks contained within the specific cache set. Notably, certain blocks within the set undergo a higher volume of writes than others, resulting in accelerated wear for these heavily utilized blocks as compared to their counterparts. Conversely, InterV denotes disparate write counts observed among cache sets, highlighting a lack of uniformity in data writing patterns across the cache sets. Consequently, this imbalance accelerates the degradation of heavily utilized sets, leading to their premature breakdown relative to their less active counterparts. To quantify WV, we use coefficients to measure how write operations are unevenly distributed across the STT-RAM LLC.

These metrics help assess their impact on performance and lifespan. InterV and IntraV are represented by Equation 2.1 and Equation 2.2, respectively, as defined in [12].

$$InterV = \frac{1}{W_{avg}} \sqrt{\frac{\sum_{i=1}^{N} (\sum_{j=1}^{M} w_{i,j}/M - W_{avg})^2}{N-1}} \tag{2.1}$$

$$IntraV = \frac{1}{W_{avg}.N} \sum_{i=1}^{N} \sqrt{\frac{\sum_{j=1}^{M} (w_{i,j} - \sum_{j=1}^{M} w_{i,j}/M)^2}{M-1}} \tag{2.2}$$

Here, $W_{i,j}$ is the block write count of i^{th} set and j^{th} way. Average write count W_{avg} is defined as:

$$W_{avg} = \frac{\sum_{i=1}^{N} \sum_{j=1}^{M} w_{i,j}}{NM} \tag{2.3}$$

M is the total number of cache ways in a set and N is the total number of cache sets. Relative lifetime (RL) is used to quantify the lifespan of the cache. It is defined as the inverse of the maximum write count (W_{max}) on a cache line in the LLC [14] as depicted by Equation 2.4. It is a derivative of raw error tolerant lifetime which is the time taken by the cache for the first error to occur in a cache line after specific number of writes to that cache line. The term "raw" implies that no error recovery mechanism is considered.

$$RL = \frac{1}{W_{max}} \tag{2.4}$$

Cache wear-leveling seeks to address both InterV and IntraV by distributing write operations more evenly across all the cache lines. Its goal is to minimize the maximum write count on any single cache line, thereby extending the overall lifespan. Therefore, Relative Lifetime Improvement (RLI) can be defined as:

$$RLI = \frac{RL_{Improvement}}{RL_{Baseline}} \tag{2.5}$$

RLI is the improved technique's lifetime with respect to the baseline LRU. The values of RLI are calculated to be in times.

2.3 Techniques to Improve Lifetime in STT-RAM based LLCs

The RLI of the cache can be increased with reduction in the maximum write count in any specific block in the cache. In order to significantly extend the lifetime of STT-RAM LLC, researchers have identified three principal types of solutions:

1. Write Variation Leveling (Wear Leveling)
2. Write Bypassing (Write Reduction)
3. Hybrid Caches

2.3.1 Wear Leveling Techniques

The main idea of any wear leveling technique is to spread the writes across the STT-RAM LLC as STT-RAM cells have limited endurance, as repeated writes to a specific STT-RAM cell can cause the cell to wear out and loose its lifespan. In the case of NVMs, limited write endurance is a significant concern. The lifetime of the NVM based LLC can be extended by decreasing the WVs both within and across the sets. Majority of these techniques aims to reduce the write variation across the LLC in the form of InterV or IntraV. Therefore, Wear leveling techniques have been applied at various levels of granularity. These include techniques at:

a) **set-level granularity** : Reduce InterV

b) **way-level granularity** : Reduce IntraV

c) **block-level granularity** : Reduce both InterV and IntraV

Majority of the state-of-the-art techniques that have been proposed are at the set-level granularity or way-level granularity of wear leveling. Wang et al. introduced i^{2}WAP [12] that is an amalgamation of two different techniques: PoLF and SWS. It aims to reduce both InterV and IntraV. PoLF (Probabilistic Set Line Flush) was introduced to reduce the IntraV in a specific cache in order to improve the lifetime. It implements write blocking to a designated block within the cache set. It designates a block as invalid following a predefined number of writes, known as the Flush Threshold (FT). Should the number of writes to the block exceed the FT, it is blocked for further write operations and that specific block is deemed as invalid. SWS (Swap Shift), also a part of i^{2}WAP is aimed at reducing the InterV. It employs data invalidation during remapping of cache set address. The objective of the SwS scheme is to modify the mapping of cache physical sets by rotating the stored data across them. However, simultaneously shifting all cache sets results in substantial performance overhead. To mitigate this issue, SWS only exchanges the mapping of two sets at a time, ensuring that over several swaps, all cache sets are gradually shifted. It employs a global write counter for the whole cache and after a certain threshold is reached two of the sets are shifted and the counter is reset. Once again when the global write counter is saturated then the shift takes place. On swapping the data and set ids of the sets are interchanged.
Sequoia [15] proposed an amalgamation of two different techniques, namely, Writeback Aware Displacement (WAD) and On-access inter-set Swapping (OAS). It is effective in reducing both the IntraV and InterV. WAD proposes an approach for identifying and managing blocks within a set that experiences significant write traffic, utilizing a sophisticated counter mechanism. There is no use of extra counters as the lower bits of global counters are considered as set wise write counter. If any write hit is incoming to a set saturated with a saturated counter it is considered as a hot line and that hot line is invalidated and no writes that cache line are allowed. In case we apply the clean LRU process, the write-intensive blocks are subsequently exchanged with a victim block

within the identical set, including a process to mitigate any substantial increase in the LLC miss ratio. OAS proposes a set remapping technique where the mapping of write hot sets are swapped with those sets that are not write hot. The operation of OAS relies on a selection algorithm that identifies hot and cold sets, along with a swapping algorithm that determines the appropriate line within the selected set for swapping. This process facilitates data movement between the two sets. The mapping can either be changed individually or in a grouped manner.

Agarwal et al. [14] propose three distinct methodologies: Static Window Write Restriction (SWWR), Dynamic Window Write Restriction (DWWR), and Dynamic Way Aware Write Restriction (DWAWR). In SWWR, the cache is logically partitioned into T equal-sized windows, each containing an identical number of ways. During execution, one window is designated as a write-restricted window for a predetermined epoch, during that epoch any incoming write to any specific way in that window will not be allowed and that write will be redirected to the other ways. The window to be placed under the restriction is determined in a round robin manner. In DWWR also, the cache is logically partitioned into T equal-sized windows, each containing an identical number of ways. The write-restricted window is not predetermined but changed after every epoch based on the write intensity in the specific window. If during a specific epoch a window has the highest write intensity, then that window is placed under write restriction for the next epoch. In order to determine the write intensity in a specific epoch each window is equipped with a write counter that is responsible for keeping count of the number of writes in that specific window per epoch, after every epoch the write counter is reset again to 0. In contrast, for DWAWR a specific number of write hot ways are selected and are write-restricted for a predetermined interval or epoch. These specific write hot ways are selected dynamically and they change every epoch. Each way has a write counter that is responsible for counting the writes into that specific way and is reset after every epoch.

Mittal et al. proposed WriteSmoothing [17] that aims to extend the cache lifetime by reducing both the IntraV and InterV. Each LLC is divided into multiple modules and the main aim of the technique is to reduce the WV across the modules of the cache. After every epoch a write hot module is determined and future writes to that write hot module is restricted and redirected to the modules that are not write hot.

EqualChance [16] was proposed to mitigate IntraV in order to increase the lifetime of the system. EqualChance tracks the number of writes to each set and periodically shifts a write-intensive data item to a block in a lower position within the LRU stack. This block is presumed to have experienced fewer writes during the recent execution interval. By redirecting future writes from a hot (frequently written) block to a cold block, this approach aids in achieving wear leveling.

Mittal et al. proposed LastingNVCache [18], which operates on the key principle that by periodically flushing a frequently written data item without updating its LRU age information, it will be reloaded into a cold block within the set. This mechanism allows future writes to the data item to be redirected from a hot block to a cold one, promoting

IntraV reduction. By evenly distributing write pressure across blocks, the worst-case write load on any single block is reduced, ultimately extending the cache's lifetime.

Mittal et al. introduced a technique that employs color mapping [34]. This report presents a wear-leveling technique for LLCs designed with non-volatile memory devices. The technique utilizes a cache coloring scheme to introduce a software-controlled mapping layer between groups of physical pages, known as memory regions, and cache sets. Periodic computation of write counts to various cache colors informs adjustments in the mapping of selected colors, directing write traffic to the least utilized cache colors. This approach promotes wear leveling. While both InterV and IntraV may arise in the cache, the technique primarily mitigates InterV and can complement other methods aimed at addressing IntraV.

Mittal et al. [35] introduces EqualWrites, a technique aimed at extending cache lifetime by reducing IntraV. The core idea behind EqualWrites is that when the difference in the number of writes between two blocks within a cache set exceeds a defined threshold (denoted as Ω), it signals significant IntraV. To mitigate this, the data and addresses in these blocks are swapped, redirecting future writes from the hot block to the cold block, thereby promoting wear leveling.

Agarwal et al. [36] proposes a technique that utilizes Dynamic Associativity Management (DAM) to facilitate wear leveling. The cache is divided into groups of sets, known as fellow groups. Each cache set is partitioned into a Normal (NP) and Reserve (RP) section. The NP section operates like a traditional cache, while the RP section is shared across all sets within a fellow group. When a set experiences high write traffic, it can utilize the RP space of other sets within the group, enabling dynamic associativity management. This approach does not aim to increase associativity but instead uses the RP and fellow sets to achieve wear leveling. Write redirection is determined by set-level write counters, directing excess writes from heavily used sets to the RP sections of lightly written ones. Agarwal et al. [37] proposed two different techniques: Fellow Set with Static Reserve Part (FSSRP) and Fellow Set with Dynamic Reserve Part (FSDRP). FSSRP works similar to [36] where cache sets are divided into groups, referred to as fellow groups, with each set belonging to either of the two sections: Normal and Reserve. Within a fellow group, sets can access the Reserve sections of other sets, enabling uniform distribution of writes across the group. FSDRP on the other hands works similar to DWWR [14]. In addition to FSSRP, the cache is vertically divided into multiple windows. During execution, a different window is designated as the reserve section over a specific interval, allowing for uniform distribution of writes.

Soltani et al. [38], proposed a cluster based set mapping technique where the cache is organized into clusters, each comprising multiple sets. The proposed method leverages dynamic cluster virtual address mapping, which adjusts cluster virtual addresses in real time to evenly distribute write traffic across all cache sets.

Sivakumar et al. [39] proposed logical cache partitioning at the LLC level where the NVM based cache is divided into Instruction and Data parts in each set, with specific blocks being

assigned either of the two. They have proposed a Virtual Split Cache (ViSC) where the cache is split into two. This will allow simultaneous read write access in the the LLC and the logical mapping of LLC ways between instruction and data is periodically alternated to ensure uniform distribution of writes. Shivakumar et al. [40] proposed dynamic techniques over ViSC like Enhanced-ViSC (E-ViSC) which dynamically adjusts its reorganization interval based on observed writing patterns, allowing it to adapt to changing workloads and optimize performance over time. It also proposed Protean-ViSC (P-ViSC) where the partitioning is fixed to some extent. This technique [41], proposed Write Aware Last Level Non-Volatile Cache (WALL-NVC) with a replacement policy called Least Recently Used Cold Block (LRU-CB). This dual-stage wear leveling technique consists of two key components. The first stage introduces a novel LRU-CB replacement policy, designed to select more optimal victim blocks for cache replacement in NVMs. The second stage applies a conventional write distribution strategy, which operates in conjunction with LRU-CB to enhance the cache's lifespan.

2.3.2 Write Bypassing Techniques

Sparsh Mittal [42] presented a detailed survey of different cache bypassing techniques (CBT) for write or read operations. This gives us a thorough understanding of the various uses of cache bypassing and the contexts for which cache bypassing can be implemented.
Zhang et al. [19] introduced a statistic based write bypassing technique (SBAC) for asymmetric-access caches. SBAC addresses the varying costs of read and write operations to enable effective bypassing decisions. Instead of targeting individual data blocks, the method relies on the statistical behavior of data across the entire cache. This approach significantly reduces design and run-time overhead while improving decision accuracy, as the statistical behavior of data remains stable and predictable across many applications.
Ahn et al. [20] introduces the Dead Write Prediction Assisted STT-RAM Cache Architecture (DASCA) that bypasses write operations to the cache only when it predicts that doing so will not result in additional cache misses. As predicting such outcomes requires insight into future cache access patterns, DASCA incorporates a dead write predictor to determine whether a given write operation will be a dead write or not.
Kim et al. [21] proposes a novel bypass scheme for NVM-based inclusive LLC, incorporating a small tag cache called Inclusive Bypass Tag Cache (IBTC). IBTC preserves the inclusion property by storing tag information to handle cache coherence requests while considering the write endurance limitations of NVMs. To mitigate accelerated memory wear-out caused by frequent write operations, a monitoring mechanism tracks the write intensity of IBTC, preventing excessive updates. The monitor increases the intensity when IBTC is written due to bypass operations and decreases it when blocks are written into the LLC. Bypass decisions are made based on these values: If the intensity value is below zero, the block is bypassed; otherwise, it is stored in the LLC.
Dybdahl et al. [43] initially proposed a write bypassing technique for LLC performance improvement. They identify the potential misses early and tend to bypass only those

blocks. This was one of the first techniques that took a decision by taking into account the reuse distance of the block present in the cache. It led a increase in the performance of the system.

Density proposed by Korgaonkar et al. [13] highlighted a reuse based cache bypassing technique that assigns a priority to any incoming block. This helps in determining their liveness score which is important in identifying weather a write-back to a LLC from the L2 cache is having the chance to be reused again in the near future. If there is a chance to be reused then it should be placed in the LLC or else it can be bypassed directly to the main memory or stored in a small temporary buffer to be reused again in the near future. The degree of bypassing is dependent on the Write Congestion Aware Bypass (WCAB) algorithm.

Bagchi et al. [22] recently proposed, Performance Optimization and Endurance Management for Non-volatile Caches (POEM) which aimed at aggressively bypassing both write backs from the upper level cache and writes from the main memory while redistributing the remaining writes evenly among the cache lines. The bypassing is mainly done based on the write hotness of that location and the reuse distance of the incoming block. This not only helped them in gaining a significant improvement in performance but also helped with wear leveling and thereby enhancing the lifetime of the NVM based LLC.

An obstruction-aware cache management policy for STT-RAM last-level caches (OAP) is proposed by Wang et al. [44], where on any obstruction in the read/write queue of the LLC any (request either read or write) can be bypassed or managed accordingly. This approach significantly enhances performance and reduces energy consumption across various workloads, all while introducing minimal hardware overhead.

2.3.3 Hybrid Caches

In order to integrate the best of SRAM and STT-RAM technologies for LLC's a lot of hybrid cache architectures were proposed that aimed at providing the speed and latency of the SRAM's with the density and nonvolatile nature of STT-RAM's.

Jog et al. proposed Cache Revive [24] a STT-RAM LLC with varied retention timed cells. Here, different portions of the cache have different retention time STT-RAM cells. The trade-off for lower retention time is higher endurance in case of STT-RAM cells. Therefore, the areas having lower retention time cells are considered as the write intensive areas and writes are migrated to those portions.

Mittal et al. proposed Ayush [26] which is a combination of both SRAM and STT-RAM in the LLC. It relies on migration of write intensive blocks to the SRAM part of the cache while keeping the read intensive blocks in the STT-RAM part. Here, few eays are designated as SRAM based ways and a few ways are designated as STT-RAM based. The writes can therefore be migrated from the STT-RAM ways to the SRAM ways within a single set. Therefore, it can enhance the lifetime of the STT-RAM LLC in this manner.

Lin et al. [45] proposed two access-aware policies: STT-RAM Write Management Policy

and SRAM Read Management Policy, that were used to address unbalanced STT-RAM wear-out and help in reducing the WV. They also proposed a dynamic partitioning scheme that adapts based on wear-out levels of the STT-RAM blocks. A novel hybrid cache architecture is proposed, incorporating SRAM banks, STT-RAM banks, and STT-RAM/SRAM hybrid banks for chip multiprocessors. This design aims to optimize performance and energy efficiency by leveraging the unique advantages of both SRAM and STT-RAM technologies within a single multiprocessor cache system. Writes are shifted from STT-RAM portion to SRAM portion and reads from SRAM portion to STT-RAM portion.

Bao et al. [27] introduces MacroTrend: A Write-Efficient Cache Algorithm for NVM-Based Read Cache. In this technique a macroscopic trend prediction method is introduced to identify long-term hot blocks by analyzing their macro trends, represented through access count histograms. Based on this MacroTrend prediction, a new cache replacement algorithm is developed to significantly reduce the number of writes while improving the cache hit ratio.

Wang et al. [46] presents a cost-effective adaptive block placement policy for hybrid LLCs (SRAM + STT-RAM), called APM (Adaptive Placement and Migration). The technique optimally places cache blocks based on their access patterns. LLC write accesses are categorized into three types: core-write, prefetch-write, and demand-write. A core-write occurs when data is written directly from the core, either as a write-through or as dirty data evicted from a write-back core cache. A prefetch-write results from an LLC replacement triggered by a prefetch miss, while a demand-write is triggered by an LLC replacement due to a demand miss. The technique leverages the insight that block placement is often initiated by a write access, and each write type's characteristics can be adapted to different block placement strategies. A low-overhead, low-complexity pattern predictor is employed to forecast access patterns for each write class, guiding the block placement process.

Bhosale et al. [47] proposed a hybrid last-level cache (LLC) architecture, called SLAM, which combines, combining SRAM and Spin-Transfer Torque Random Access Memory (STTRAM) to achieve a superior power-performance balance compared to conventional SRAM-only, STTRAM-only, and previously proposed hybrid STTRAM-SRAM LLC architectures. The SLAM framework is designed to reduce write operations to the STTRAM region of the hybrid LLC, thereby minimizing STTRAM's write energy consumption.

Wu et al. [23] proposed different memory technologies at different cache levels and also different memory technologies at the same level in order to improve the endurance. There are two types of hybrid cache architectures (HCA): inter-cache level HCA (LHCA), where different levels in the cache hierarchy are composed of distinct memory technologies, and intra-cache level or region-based HCA (RHCA), where a single cache level is partitioned into multiple regions, each utilizing a different memory technology. In the RHCA design with fast and slow regions, a hierarchical NUCA cache is proposed, featuring a centralized swap buffer, parallel address search, and LRU replacement across the cache regions. This

design enhances cache efficiency by optimizing data movement and access between the fast and slow regions.

Kuan et al. [29] proposed HALLS, a highly adaptable last-level STT-RAM cache, as a viable solution for reducing the write energy in STT-RAM-based LLCs. HALLS leverages cache configuration and retention time adaptability to optimize performance. The design features a multi-banked cache that supports dynamic configuration through bank shutdown (to adjust cache size), bank concatenation (to modify associativity), and multi-line fetch (to alter line size). Each cache bank is provisioned with different retention times to meet the diverse needs of various applications. Runtime profiling allows data blocks to be strategically placed in banks with appropriately provisioned retention times, minimizing energy consumption without significantly impacting latency.

Quan et al. [25] proposed a technique, where a prediction table-based cache line replacement and management policy (PTHCM) is introduced for a hybrid L2 cache consisting of STT-RAM and SRAM. PTHCM is based on the observation that most cache lines in L2 are accessed only a few times, and the number of writes tends to equal the number of accesses during consecutive stays in the L2 cache. Many cache lines become dead after a few accesses, reducing SRAM utilization, and are replaced when they become LRU lines, either swapped with STT-RAM or sent to memory. To optimize this process, a prediction table is added to track access information for each cache line, predicting dead lines, frequently-written lines, and less-written lines based on historical data. Using this information, PTHCM replaces dead lines in SRAM early and allocates frequently-written lines to SRAM, thereby reducing write operations to STT-RAM and improving overall hybrid cache efficiency.

Mittal et al. proposed ENLIVE [48], a technique designed to enhance the lifetime of non-volatile caches by reducing write operations. It introduces a small SRAM storage, called HotStore (e.g., 128 entries), to temporarily store frequently accessed blocks. By migrating frequently used blocks to HotStore, future accesses are served from this fast SRAM buffer, improving both performance and energy efficiency. This approach significantly reduces the number of writes to the NVM cache, thereby extending its lifespan.

2.3.4 Miscellaneous Techniques

There has been various other techniques also that aim at improving lifetime and performance. Initially, Joo et al. [49] proposed a technique that incorporated bit-level wear leveling and read before write technique for PCM based LLC in order to enhance their performance and lifetime.

Saraf et al. [30] proposed a few different refresh-aware cache replacement policies designed to prioritize the replacement of cache blocks that are nearing their expiration. The Recency-Aware Replacement (RCR) policy introduces a “Recently Accessed” (RA) bit for each cache block, enabling better replacement decisions by prioritizing recently accessed blocks, thus enhancing cache efficiency. In cases where certain blocks in a set are repeatedly

written, the Refresh-Aware Replacement (RFR) policy avoids selecting them as victim blocks, leading to increased write counts and potentially reducing endurance. RFR is proposed to prioritize endurance over performance. To combine the advantages of both strategies, a new policy called the Refresh and Recency-Aware Replacement (FCR) policy is proposed. This approach balances recency awareness with periodic refreshing, optimizing both performance and endurance in cache management.

Asadi et al. [50] brought to the limelight how to identify bit-level write patterns and reduce their activity through data manipulation. Some data patterns in workloads are observed more frequently during application execution. By analyzing these patterns, certain studies aim to reduce data variations between frequently observed patterns through coding techniques, minimizing redundant writes in consecutive operations on the same NVM cells. Building on this, an online pattern recognition technique called Wearout Informed Pattern Elimination (WIPE) is proposed to improve last-level cache endurance. WIPE avoids writing frequent data patterns by encoding incoming data blocks, or their frequent patterns, into an index referencing a table that stores these patterns. The encoded data is typically smaller than the original block, leading to a reduction in the total number of write operations in cache cells.

Sivakumar et al. [51] proposed Trace Buffer Assisted Non-volatile Memory Cache (TANC), along with its variants, to minimize the impact of repeated writes on memory cells by utilizing the Embedded Trace Buffer (ETB). TANC incorporates an ETB, a wear-leveling module, and a skip module. It leverages the unused SRAM-based ETB to handle write requests to MLC NVMs, taking advantage of SRAM's faster write speeds and lower energy consumption. This approach reduces latency and write energy while extending the lifetime of MLC NVM caches. Unlike hybrid caches, which require more area and incur higher leakage power, TANC harnesses the strengths of NVMs and utilizes unused resources like ETB for performance gains.

Priya et al. [52] introduces a compression-based wear leveling technique to address write limitations and enhance cache longevity. The proposed approach reduces the number of bit writes and ensures an even distribution across cache sets. It utilizes frequent pattern compression to minimize each word write using predefined patterns and disperses writes within words for uniform distribution. Furthermore, bit transitions are minimized by selecting cache lines with the fewest transitions. The wear leveling mechanism is employed to improve cache endurance by evenly distributing write operations.

Wang et al. [53] proposed a technique to balance write variations between the upper and lower halves of narrow-width data, by introducing two swapping schemes, Swap-on-Write (SW) and Swap-on-Replacement (SRepl). Additionally, two optimization techniques are integrated: multiple dirty bits (MDB) and read-before-write (RBW) with the word-level swapping design. To further mitigate write variation at the partition level, cache partitioning is employed to enhance cache lifetime. Noting that different applications exhibit varying cache access and write behaviors, the last-level cache is partitioned, and write variations are balanced through partition swapping.

2.3.5 Summary of this Section

Table 2.2 provides the summarized description of the various lifetime enhancement techniques discussed in this section. The first column of the table shows the different types of techniques used, the second column shows which papers have been published in correspondence with the respective techniques, and the third column indicates the advantages of these papers.

Table 2.2: Classification and Brief Overview of Lifetime Enhancement Techniques for NVM based LLC (LI=Lifetime Improvement, RedInterV=Reduction in InterV, RedIntraV=Reduction in IntraV and PI=Performance Improvement and Energy Reduction=ER).

Technique (Type)	Paper	Advantages
Wear Leveling	Wang et al. [12], 2013	RedInterV, RedIntraV and LI
	Mittal et al. [34], 2013	RedInterV and LI
	Mittal et al. [16, 17, 18], 2014	RedInterV, RedIntraV and LI
	Jokar et al. [15], 2016	RedInterV, RedIntraV and LI
	Mittal et al. [35], 2016	RedIntraV and LI
	Soltani et al. [38], 2016	RedInterV and LI
	Agarwal et al. [36], 2017	RedInterV and LI
	Agarwal et al. [14, 37], 2019	RedIntraV and LI
	Sivakumar et al. [39], 2021	RedInterV and LI
	Sivakumar et al. [41], 2023	RedInterV, LI and PI
	Sivakumar et al. [40], 2024	RedInterV, RedIntraV, LI and PI
Write Bypassing	Dybdahl et al. [43], 2006	LI and PI
	Wang et al. [44], 2013	LI and PI
	Zhang et al. [19], 2014	LI
	Ahn et al. [20], 2014	LI and PI
	Kim et al. [21], 2015	LI and PI
	Korgaonkar et al. [13], 2018	LI and PI
	Bagchi et al. [22], 2024	RedInterV, RedIntraV, LI and PI
Hybrid Cache	Wu et al. [23], 2009	LI
	Jog et al. [24], 2012	LI, PI and ER
	Quan et al. [25], 2012	LI, PI and ER
	Wang et al. [46], 2014	LI, PI and ER
	Mittal et al. [26], 2015	LI and ER
	Lin et al. [45], 2015	LI, PI and ER
	Mittal et al. [48], 2016	LI, PI and ER
	Kuan et al. [29], 2019	LI, PI and ER
	Bhosale et al. [47], 2019	LI and ER
	Bao et al. [27], 2022	LI and ER
Miscellaneous	Joo et al. [49], 2010	LI and ER
	Asadi et al. [50], 2017	LI
	Wang et al. [53], 2017	LI
	Saraf et al. [30], 2019	LI
	Priya et al. [52], 2023	LI
	Sivakumar et al. [51], 2024	LI and PI

Through thorough analysis on the various types of techniques that have been implemented to deal with the issues caused by the write operations in these areas, we have identified that most of the works tend to solve only one of the three major issues caused by writes in an STT-RAM. There are hardly any works that have been able to solve two or all three problems concerning these writes. Therefore, this provides us a window of opportunity to propose different state-of-the-art techniques that tend to solve various problems concerning the writes together.

2.4 Attacks on STT-RAM LLCs

This section deals with various types of attacks that plague the implementation of STT-RAM as LLCs. It delves into details of the research conducted on attacking the caches.

2.4.1 Cache Timing Channel Attacks

Cache timing channel attacks can be categorized based on their underlying techniques. There are 2 different types of cache timing channel attacks namely: Cache Side Channel Attacks (SCA) and Cache Covert Channel Attacks (CCA).

2.4.1.1 Cache Side Channel Attacks (SCA)

A cache SCA an attacker takes advantage of the shared nature of the LLC between different processes or users. By monitoring changes in cache access times or analyzing cache eviction patterns, the attacker can deduce what data or instructions the victim is accessing, which may lead to leakage of sensitive information (such as encrypted keys).
Irazoqui et al. [54] introduced a shared cache attack designed to operate across multiple cores, specifically targeting the shared L3 cache in a (P+P) attack variant.
Moghimi et al. [55] introduced Cache Zoom that is capable of virtually tracking all memory accesses of SGX enclaves with high spatial and temporal precision from the LLC by exploiting access patterns. Till now all these attacks have been proposed on SRAM based LLCs to track the SGX enclaves that are responsible for providing a secure DRAM functionality, mainly because STT-RAM based LLCs were not a viable alternative, but with modern solutions provided to make it a possible LLC alternative these attacks will have its effectiveness on STT-RAM based LLCs as well.
Zhang et al. [56] introduced this paper that presents an access-driven side-channel attack that allows a malicious virtual machine (VM) to extract fine-grained information from a victim VM on the same physical host.

2.4.1.2 Cache Covert Channel Attacks (CCA)

A cache CCA is a method where malicious processes, like a spy and a Trojan, secretly exchange information by exploiting the shared cache memory of a computer system. This type of attack utilizes cache timing channels to enable covert communication between processes without direct contact, thereby evading the security measures designed to prevent such interactions.
Kaur et al. [57] proposed a modified cache-collision attack based on P+P that can create a covert channel utilizing the dynamic cache partitioning applied to the LLC. These types of attacks require careful mitigation in modern processors.
Maurice et al. [58] proposed C5, where a covert channel is demonstrated across virtual machines on modern hardware, effectively addressing the addressing uncertainty that has

hindered previous covert channels.
Yao et al. [59] proposed a technique which uncovers a vulnerability in cache coherence protocols, allowing adversaries to manipulate cache block states to alter access timing and covertly communicate secrets.

2.4.2 Cache Contention Attacks

Irazoqui et al. [54] introduces a fine-grained cross-core cache attack that leverages access time variations in the LLC.
Disselkoen et al. [60] introduces an alternative cache attack mechanism, PRIME+ABORT, which does not rely on timing differences or timed operations. Instead, it exploits Intel's Hardware Transactional Memory (TSX).
Oren et al. [61] present a micro-architectural side channel attack that operates entirely within a web browser, without requiring the attacker to install any software on the victim's machine.

2.4.3 Cache Occupancy Attacks

Chakraborty et al. [62] indicates that most existing research focuses primarily on protecting the LLC from contention-based cache attacks. This work explores alternative cache attack variants and assesses how secure cache design principles influence these attacks [63, 60].
Shusterman et al. [64, 65] explored cache side-channel attacks using a novel model in which an adversary sends malicious JavaScript to the target user's computer, exploiting cache contention effects to identify other visited websites.

2.4.4 Cache Rowhammer based Attacks

Khan et al. [66] explores the effects of Row Hammering on STT-RAM by exploiting its write vulnerabilities. STT-RAM's high write current and long latency can induce ground bounce, whose magnitude depends on the data being written and can be spread to adjoining cells in a highly dense environment.
Staudigl et al. [67] highlighted that the advantages of NVM's in LLCs and introduce new security vulnerabilities, such as the NeuroHammer [68] attack, which allows adversaries to intentionally flip bits in ReRAM.

2.4.5 Cache Miscellaneous Attacks

Quereshi [69] introduced two new attacks that significantly advance the state-of-the-art in forming eviction sets. This attack requires increasing CEASER's [70] remap rate to 35%. The second attack leverages different cache replacement policies (such as LRU, RRIP, and Random) to quickly form eviction sets, which requires an impractical increase in remap rate to over 100%. To counter these attacks, the paper proposes Skewed-CEASER (CEASER-S), which enhances robustness by partitioning cache ways and mapping cache lines to different sets in each partition.

Gruss et al. [71] introduce cache template attacks, a generic technique that automatically profiles and exploits cache-based information leakage from any program, without requiring prior knowledge of software versions or system details.

2.4.6 Summary of this Section

Table 2.3 provides the summarized description of the various kinds of attacks discussed in this section their targets and different environments in which they are usually run on.

Table 2.3: Classification of LLC Attacks.

Attack Type	Paper	Target	Environment
SCA	Brasser et al. [72], 2017	RSA Genomic Processing	Native
	Schwarz et al. [73], 2017	RSA	Cloud
	Sinan et al. [74], 2016	RSA	Cloud
	Götzfried et al. [75], 2017	AES	Cloud
	Disselkoen et al. [60], 2017	AES	Cloud
	Moghimi et al. [55], 2017	AES	Cloud
	Irazoqui et al. [54], 2015	AES	Cloud
	Gullasch et al. [76], 2011	AES	Native
	Liu et al. [77], 2015	ElGamal	Native
	Oren et al. [61], 2015	user‘s privacy	Native+Cloud
	Aciiçmez et al. [78], 2010	DSA	Native
	Wang et al. [79], 2019	NA	Native
	Gruss et al. [80], 2016	AES, keystoke timing	Native
	Irazoqui et al. [81], 2014	AES	Virtual
	Gangwar et al. [82], 2024	AES	Native
	Gülmezoğlu et al. [83], 2015	AES	Cloud
	Allan et al. [84], 2016	ECDSA	Native
	Hornby et al. [85], 2016	users privacy	Native
	Yan et al. [86], 2019	RSA	Native
	Jiang et al. [87], 2017	AES	Native
CCA	Liu et al.[77], 2015	75KBps	Virtual
	Maurice et al. [58], 2015	1291bps(Native), 751bps(Virtual)	Native+Virtual
	Watson et al. [88], 2009	0.025bps	Cloud
	Yan et al. [89], 2019	0.2Mbit/s	Native
	Maurice et al. [90], 2017	75KBps(Native), 36KBps(Virtual)	Native+Virtual
	Percival et al. [91], 2005	400KBps	Native
	Xu et al. [92], 2011	262.47bps(Native), 223.71(Virtual)	Native+Virtual
	Wu et al. [93], 2012	23.8KBps	Native
	Yao et al. [59], 2019	800KBps	Native
	Gruss et al. [80], 2016	NA	Native
	Irazoqui et al. [94], 2016	AES, El gamal	Native
Cache Contention Attacks	Irazoqui et al. [54], 2015	AES	Cloud
	Disselkoen et al. [60], 2017	AES	Cloud
	Oren et al. [61], 2015	user‘s privacy	Native+Cloud
Cache Occupancy Attacks	Chakraborty et al. [62], 2023	Cache Overfill	Native
	Shusterman et al. [64], 2019	Website	Browser
	Shusterman et al. [65], 2021	Website	Browser
Row Hammer Attacks	Khan et al. [66], 2018	Cache Location	Native
	Staudigl et al. [67], 2024	Cache Location	Native
Miscellaneous Attacks	Quereshi [69], 2019	Cache Location	Encrypted Cache
	Gruss et al. [71], 2015	User‘s information	Native

Despite a lot of research carried out in this area. STT-RAM based LLCs can still be vulnerable if a single or multiple locations are targeted and written into continuously over a period of time. These repeated writes will cause rapid degradation of the cache line and reduce the lifetime of the STT-RAM LLC. Therefore, there is an opportunity to mold an attack that can target specific cache locations to write into continuously in order to degrade its lifetime.

2.5 Summary of this Chapter

The initial section of this chapter provided the essential background information required to comprehend the configuration of the STT-RAM LLC. Thereafter, we dived deeper into how writes effect the the STT-RAM LLC and its dependencies on write variation. We investigated various techniques that tried to reduce the adverse effects caused by the writes. The second section of the chapter offers a detailed review of the existing literature on methods to mitigate the adverse effects of writes like enhancing the lifetime of STT-RAM LLC, reducing the write latency and reducing the write energy. The third part offers a detailed review of existing attacks on LLCs and shows how the targeted writes can cause vulnerability in the lifetime of STT-RAM based LLCs.

Chapter 3

Dynamic Write Bypassing for Lifetime Improvement in STT-RAM LLCs

In this chapter, we have proposed a novel write bypassing technique, that bypasses writes based on the probability of reuse of an incoming block from L2 to LLC and the set/way write intensity in the STT-RAM LLC. In this technique, a set-wise/way-wise write counter is implemented in order to count the number of writes per set/way. Writes are bypassed from those sets/ways having a high write counter value. The bypass can be made into the MM or a small additional SRAM cache placed between the LLC and the MM.

Publications from this Chapter

- **Prabuddha Sinha**, Krishna Pratik BV, Shirshendu Das and Venkata Kalyan Tavva, "PROLONG: Priority based Write Bypassing Technique for Longer Lifetime in STT-RAM based LLC", in 10th International Symposium on Memory Systems (MEMSYS), Washington DC, USA, 2024. [DOI: https://dl.acm.org/doi/10.1145/3695794.3695803]
- **Prabuddha Sinha**, Krishna Pratik BV, Shirshendu Das and Venkata Kalyan Tavva, "Hide-N-Seek: Hiding Writes in Buffer for Lifetime Improvement in STT-RAM based LLC", at HiPC 2023 Student Research Symposium, in Proceedings of the 2023 IEEE 30th International Conference on High Performance Computing, Data and Analytics Workshop (HiPCW), Goa, India, 2023, pp-84. [DOI: https://doi.org/10.1109/HiPCW61695.2023.00021]
- **Prabuddha Sinha**, Krishna Pratik BV, Shirshendu Das and Venkata Kalyan Tavva, "LiveWay: Dynamic Write Bypassing for Lifetime Enhancement in STT-RAM LLC", at HiPC 2024 Student Research Symposium, in Proceedings of the 2024 IEEE 31st International Conference on High Performance Computing, Data and Analytics Workshop (HiPCW), Bangalore, India, 2024. [DOI: https://doi.ieeecomputersociety.org/10.1109/HiPCW63042.2024.00035]

3.1 Introduction

As discussed in Chapter 2, uneven write distribution is a major issue that is responsible for diminishing the lifetime of a STT-RAM LLC. When applications are run on various cores of a multiprocessor simultaneously, because of their inherent behavior, they then create WV throughout the LLC that results in drastic drop in the lifetime of the STT-RAM based

LLC. These fluctuations in write behavior not only shorten the lifespan of the STT-RAM LLC but also contribute to a reduction in LLC capacity over time. The lifetime of the STT-RAM LLC is extended with a decrease in WV both InterV and IntraV. The quantity of writes is undoubtedly another significant factor that has a direct impact on the lifetime of STT-RAM LLC. Implementing any wear leveling technique that aims to reduce InterV and and IntraV comes with its own sets of challenges.

3.2 Motivation

Figure 3.1 and Figure 3.2 illustrate the reduction in WV and the normalized IPC reduction, respectively, for various state-of-the-art wear leveling techniques as compared to the baseline. Here, the baseline is considered as STT-RAM LLC with an LRU replacement policy and no wear leveling technique. The details of the experimental setup are discussed in Section 3.5.

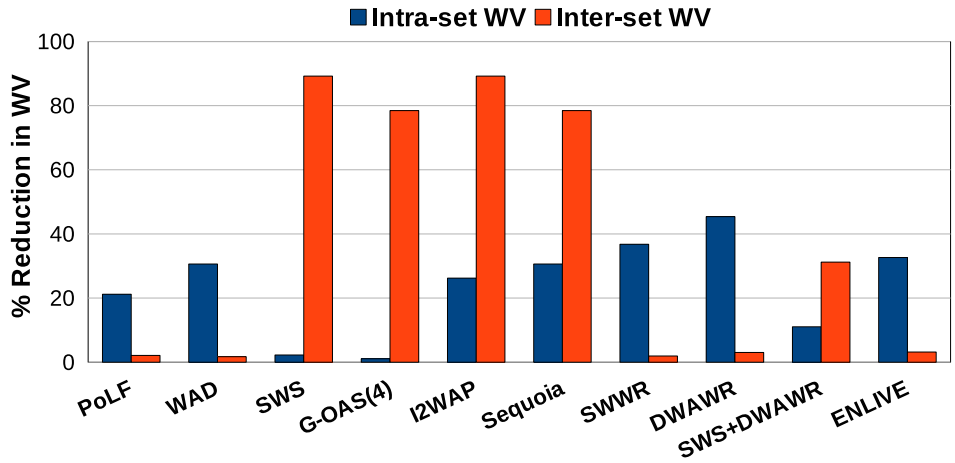

Figure 3.1: Reduction in InterV and IntraV for various state-of-the-art wear leveling techniques.

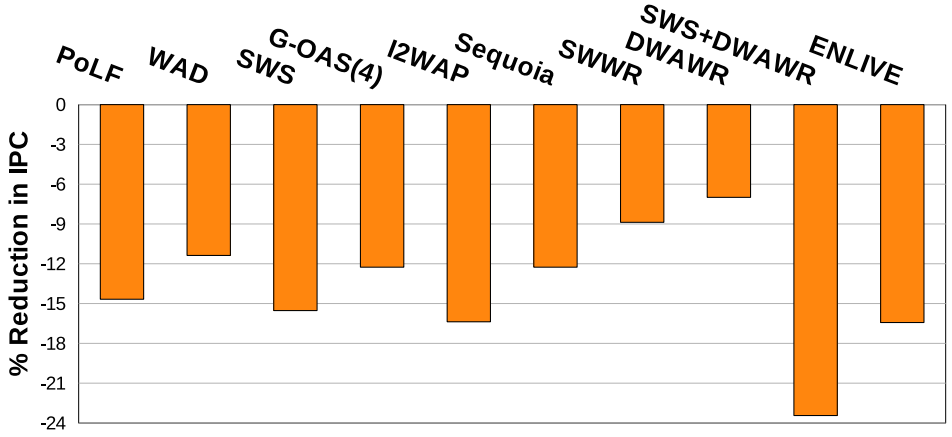

Figure 3.2: IPC Degradation for various state-of-the-art wear leveling techniques.

From the Figure 3.1, it is evident that techniques such as SWWR [14], DWAWR [14],

PoLF [12], WAD [15], and ENLIVE [48] are unable to decrease the InterV to the same extent as IntraV, while techniques such as SwS and G-OAS is able to decrease only the InterV WV. Furthermore, most of these existing techniques experience a drop in performance compared to the baseline. Figure 3.2 illustrates that there is a decline in their performance ranging from 7% to 23%. We combine the best of both the best techniques: SwS + DWAWR and have discovered that it did not perform as expected. The execution of SwS and DWAWR are not conductive as they interfere with each other's smooth functioning. The reduction in both InterV and IntraV was significantly lesser compared to when the techniques were applied individually. The IPC was also reduced significantly. SwS always leads to additional remapping. The blocking of writes into ways due to DWAWR can cause a further increase in the remapping and migration, which thereby increases the write count of the LLC. The existing wear-leveling techniques are insufficient in effectively minimizing the InterV and IntraV of STT-RAM LLC. These techniques can enhance the lifetime of a STT-RAM LLC by a maximum of up to 9 times. Through analysis of the effect of write variation, we performed experiments utilizing a simulator for a quad-core system equipped with a 16 MB 16-way set-associative STT-RAM LLC lacking wear leveling support with standard LRU replacement policy. We have used *Mix 2* shown in Table 3.6. The configuration is executed for 250 Million instructions. Figure 3.3 shows us the non-uniform write distribution across cache sets that are prevalent. It depicts the write count after every 100 sets. Non-uniform writes to an STT-RAM LLC is the major reason for decreasing endurance as well as the lifetime of the LLC, thereby inducing errors. The lifetime of the LLC depends directly on the maximum write count. The maximum write count among all the sets is 3516 while the write average is only 564. This clearly shows the huge disparity in write counts that is present among the sets. If the InterV is removed, then the overall lifetime will be enhanced by 6.23 times in this case. The problem of low write endurance is also enhanced by non-uniform write distribution across cache ways as depicted in Figure 3.4. It depicts the write count for all the ways in the cache. It can be clearly seen that a few ways have a lot of writes while some ways have very few writes. The maximum write count in one way is 16131, while the minimum write count is 1016. To this end, this technique is segregated into 2 different parts that having different objectives: A) To reduce InterV. B) To reduce IntraV.

3.3 PartA (To reduce InterV): PROLONG: Priority based Write Bypassing Technique for Longer Lifetime in STT-RAM based LLC

3.3.1 Proposed Architecture

In order to support PROLONG we propose a few architectural changes as compared to the traditional memory hierarchy. We introduce three new components, namely, the writes set counter (WSC), a small SRAM buffer located between the STT-RAM LLC and the

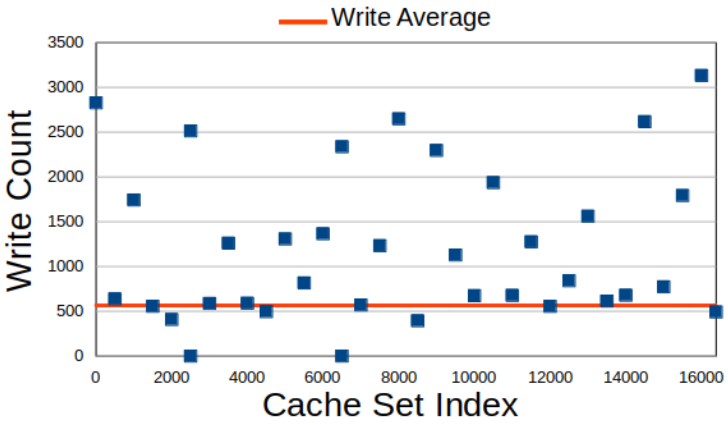

Figure 3.3: Non-uniform write distribution across cache sets

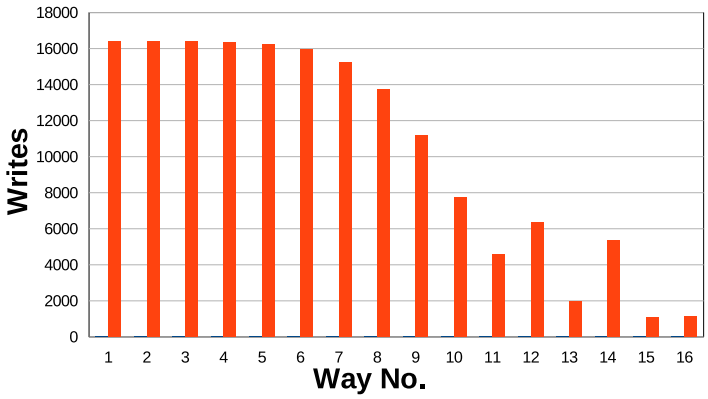

Figure 3.4: Non-uniform write distribution across cache ways

Main Memory, and a Liveness Score Counter (LSC). Figure 3.6 depicts the newly proposed architecture. RQ represents the Read queue and WQ represents the Write queue of the LLC. The SRAM Buffer is placed in between the LLC and MM in the memory hierarchy and before checking for any block in the MM we check for it in the SRAM Buffer. Each block present in the LLC is also equipped with the "Liveness Score" counter (LSC) while each set is equipped with a WSC.

3.3.1.1 Writes set counter (WSC)

Each set of the STT-RAM LLC is equipped with a dedicated 8-bit write counter. These write counters are updated whenever there is a write into that specific set of the STT-RAM LLC. Whenever the write counter is saturated in a specific epoch, for the next epoch the write count for that specific write counter is halved. This way any new incoming writes can again be counted in that counter. This way the significance of the write intensity in that counter is not explicitly dismissed for the upcoming epochs.

3.3.1.2 SRAM Buffer

A small SRAM buffer is placed in between the STT-RAM LLC and the Main Memory. This buffer is used to store selective high priority writes from write hot ways, that have been bypassed from the STT-RAM LLC by PROLONG to reduce the InterV. The writes that are stored in the SRAM buffer are those writes that have high "liveness score", i.e. these writes have a high probability to be recalled and may be used again in the near future by the LLC. We conduct studies with buffer size varying from 32KB to 512KB. The buffer is divided into multiple parts of 32 entries and a block can be mapped into a fixed part like set-associative cache. The FIFO replacement policy is used for each part separately. Prior to sending any LLC miss request to the main memory, the block is first searched in the buffer. In case of a hit in the buffer, the block is moved back into the LLC. The main purpose of this buffer is to reduce the side effects of bypassing in PROLONG and maintain the performance of the system as time taken to service a block from memory will be drastically greater than the time taken to service it from the small SRAM buffer in case of a hit.

3.3.1.3 Liveness Score Counter (LSC)

Every block in the STT-RAM LLC is equipped with a 2-bit "liveness score" counter (LSC) whose value is updated upon every access to the blocks from L2. It is used to store the liveness score of the block.

A **liveness score** represents the probability that a cache line associated with a write request will be accessed again. The cache controller may buffer a liveness score with each write request stored in the write queue. Upon detecting write congestion, such as when the write queue is full and the read queue exceeds a threshold number of read requests, the cache controller may drop the write requests with the lowest liveness scores from the write queue, thereby bypassing those write requests. The threshold number depends on NVM-LLC write latency, the turn-around time for fetching a cache line from main memory, and potentially other system configurations or parameters. This threshold may be further refined through experimental adjustments to enhance performance. The liveness score differs from known dead block predictors that bypass dead cache lines, which are used solely to improve hit rates in the LLC. A dead line is a cache line that will not be used again anytime soon. A dead block predictor may determine a liveness score or reuse distance based on usage history or the behavior of cache lines originating from the same program counter (PC). However, the present architecture focuses on decreasing the NVM-LLC write congestion while minimizing the impact on LLC hit rates. By adopting a more aggressive approach to bypassing write requests, even at the expense of some hit rate loss, write congestion can be significantly reduced, leading to a corresponding improvement in the performance of NVM-LLC devices.

Liveness Score of a block is defined as the reuse probability of the LLC block in the L2 cache. 32 observer sets present in both the L2 and the LLC are used in order for learning

the reuse probability of the block similar to [95], [13], [96]. It uses program counters (PC's) to sample the blocks. At the L2 cache, hashed PC's (Instruction addresses that last accessed the cache line in the L2) is maintained as a reference table. This table consists of four 10-bit counters each mapping to a liveness buckets ranging from 0 to 20%, 21 to 50%, 51 to 70%, and 71 to 100%. The proportion of writes in an access PC being recalled from the LLC is defined as liveness. 20% liveness implies there were 20 reads among 100 writes to the LLC in a single PC based access. Liveness counters are decremented during eviction of a block from an L2 observer set. The sampler technique described in [95], [13] provides us a reference for usage of partial-PC tags for blocks in the observer sets. Initially whenever a block is written into L2 cache the liveness counters are set as 0. Upon a LLC hit, the liveness counters of the access PC are incremented. Eviction and hits have different changes in the liveness counters for observer sets. Eviction of a block having 21-50% liveness counter leads to counter decrement of 2 and on a hit it is incremented by 10 for both 21-50% counter and 0-20% counter. A positive value of this counter indicates that it has at least 20% liveness. A 2-bit liveness score value is also assigned to every new incoming block in the L2 if it is not in the observer set, based on its previous PC based observer set value. The highest liveness bucket has the highest priority. For example, if both 71 to 100% and 51 to 70% liveness counters are both positive, the line is assigned the 71 to 100% live score. Only four buckets are being tracked for each block, hence 2 bits are required for the liveness score in L2. The liveness score is part of the L2 evicted block. An LSC is assigned to store this liveness score value in LLC. Initially, any new write will have a default liveness score of 0 corresponding to the 0-20% liveness bucket.

3.3.2 Working of PROLONG

The main idea is to selectively enable bypass for only the heavily written sets. This helps in making the writes uniform throughout the cache sets. The write bypass depends on two parameters: the liveness score counter (LSC) and the write set counter (WSC).
The LSC of any block in the LLC can range from 0 to 3. Table 3.1 represents how the write liveness score buckets and their status bits are assigned based on their liveness percentage.

Table 3.1: Bucket Distribution among sets based on Liveness Score (LS).

LSC Bucket	LSC Bit Value	Liveness Percentage
LS0	00	0-20%
LS1	01	21-50%
LS2	10	51-70%
LS3	11	71-100%

Write Hot (WH) bucket selection among sets: Whenever there is an incoming write, that can either be a L2 write-back or a Last Level Cache miss, the write set counter is incremented. The WSC indicates the write hotness of the set. A cache-set is set to be write hot if it has garnered repeated writes to it within an epoch. After every epoch, which is a certain number of instructions, we identify the write hot sets and then we

target priority based write bypassing from these write hot sets. These write hot sets are further classified into three categories equally and different degrees of bypassing are applied to each category. The number of sets selected to be bypassed is based on the bypass aggressiveness percentage. We consider bypass aggressiveness from 2% heavily written sets to 15% heavily written sets. The chosen hot sets are subsequently distributed evenly among three buckets, determined by their respective write counters. Sets with the highest write count are allocated to bucket H_3, those with a medium write count are assigned to bucket H_2, and sets with the lowest write hotness among the hot sets are placed in bucket H_1. Remaining sets that show less write activity, are allocated to H_0 bucket. To store the bucket status bit for each block, 2 bits per cache-set are required. Table 3.2 shows how the write hot buckets and their status bits are assigned.

Table 3.2: Bucket Distribution among sets based on Write Counter.

WH Bucket	**Bit Value**	**Set Status**
H_0	00	Sets that are not write hot
H_1	01	Write hot sets with low writes
H_2	10	Write hot sets with medium writes
H_3	11	Write hot sets with high writes

Bypassing decision and aggressiveness: At the beginning of each epoch, adjustments to the write counters and write hot buckets for every set are necessary. Write hot sets among all sets in an LLC are identified based on the level of bypassing aggressiveness. The more aggressive the bypassing, the greater the number of write hot sets selected. These write hot sets are divided into write hot buckets and the bucket values for each set are assigned accordingly. The distribution of write hot sets is allocated uniformly among the three write hot buckets: H_1, H_2 and H_3 as depicted in Table 3.2.

Bypass Aggressiveness is defined as percentage of sets considered as hot sets. Suppose if the write bypassing aggressiveness is 15%, we consider 15% of all sets in the cache as write hot. The top 5% write hot sets are assigned to H3 then the medium 5% is assigned to H2 and the lowest 5% among the hot sets is assigned to H1. The LSC of the blocks is automatically re-calibrated after every cache access. When a write-back request is generated from L2, the block needs to be either written in LLC or bypassed. The decision is taken based on the liveness score of the block and the WH bucket it maps to. It first inspects the WH bucket of the set to which the write block needs to be written. If it is in H_0, there will be no write bypassing. Conversely, if the writes are identified to be directed to the write hot sets (H_3, H_2 or H_1), we need to assess whether the incoming writes should be bypassed or not. Write Bypassing logic for incoming write-backs to the STT-RAM LLC sets are depicted through Table 3.3. The major rationale behind this logic was the nature of the write-backs from L2 cache, as depicted in Figure 3.5.

From Figure 3.5 it can be clearly seen that H3 has the highest write intensity making it write hot. H0 even makes up of about 85% of all the sets in the STT-RAM based LLC but consists of those sets that are ot write hot and are not effected therefore there is no need to bypass anything from H0. The write hot sets are equally divided among H1, H2,

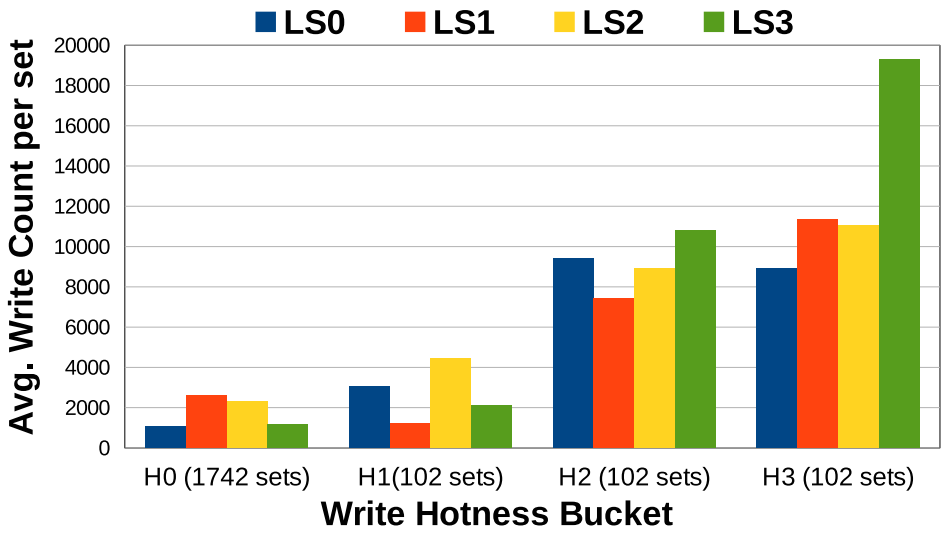

Figure 3.5: Average Write Distribution in STT-RAM based LLC with LRU among the Write Hot (WH) buckets (15%) with Liveness Score (LS).

and H3 with increasing order of write intensity. This has been calculated as an average for all write intensive traces with 1 billion instructions for the baseline LRU replacement policy. Table 3.3 shows us that the write bypassing is done in certain cases of sets being write hot based on low liveness score, indicating that these blocks might not be used again in the near future.

Table 3.3: Write Bypassing based on Write Hot Sets (WH) and LSC Buckets.

	WH Bucket	**LS Bucket**	**Bypassing**
1	H_0	LS0, LS1, LS2, LS3	No Bypassing
2	H_1	LS3, LS2, LS1	No Bypassing
3	H_1	LS0	Bypass to Main Memory
4	H_2	LS3, LS2	No Bypassing
5	H_2	LS1	Bypass to Buffer
6	H_2	LS0	Bypass to Main Memory
7	H_3	LS3	No Bypassing
8	H_3	LS2, LS1	Bypass to Buffer
9	H_3	LS0	Bypass to Main Memory

Table 3.3 indicates that selection 1, 2, 4 and 7 will always lead to no write bypassing from the designated set, i.e. the write will occur in its original state. This shows that blocks with higher liveness score, specifically LS3 will not be bypassed in any situation. Selection 3, 6 and 9 indicate that low priority blocks with LS0 are not needed to be stored in write hot sets and are directly bypassed to the main memory. The remaining selection 5 and 8 depict the conditions necessary in order to bypass writes from the write hot sets to the SRAM buffer.

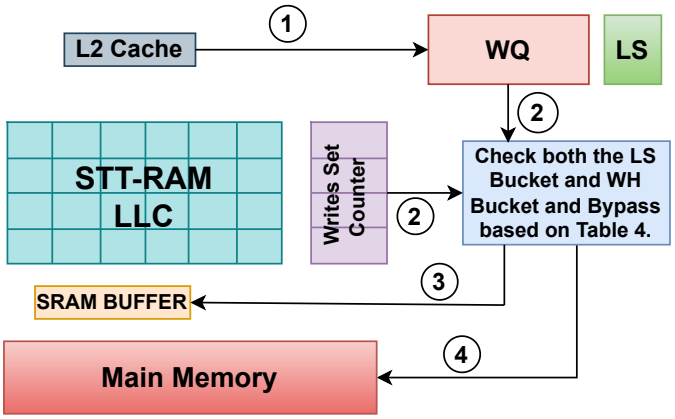

Figure 3.6: Architecture and Steps in PROLONG Algorithm.

3.3.3 PROLONG Algorithm

The work flow of the proposed PROLONG is depicted through the Figure 3.6. It comprises of the following steps:

Step-1: Upon receiving a write-back request from L2, it is inserted into the write queue (WQ) of the LLC, and its LSC value is evaluated from the incoming LS value.

Step-2: The LSC bucket value of the write request and the WH bucket value of the set are both compared and the decision of bypassing is taken based on the logic depicted in Table 3.3.

Step-3: If both conditions are met simultaneously, i.e., the LS bucket and the WH bucket are favourable, then the write may be bypassed to the SRAM buffer if they have a high LS value.

Step-4: Writes with low LS that have been selected for bypass are directly sent to the main memory.

These processes are calculated simultaneously with the working of the cache and are not under the critical path. However, the SRAM buffer access time during a LLC miss comes under the critical path. We have considered this buffer access time in our experimental analysis (Section 3.6.3).

3.4 PartB (To reduce IntraV): LiveWay: Dynamic Write Bypassing for Lifetime Enhancement in STT-RAM LLC

3.4.1 Proposed Architecture

Figure 3.7 shows our new memory hierarchy setup. In order to handle the uneven distribution of writes we introduce ***Way Wise Write Counters (WWC)***. Each counter keeps track of the writes in its respective ways. These write counters will help us keep track of write hot ways over time. Each block also has a 2-bit ***Liveness Score Counter (LSC)***. We also introduce a small SRAM buffer that is located in between the STT-RAM

LLC and the main memory. This SRAM buffer is used to store those bypassed writes that have high future reuse probability, i.e. those blocks that have a high LSC.

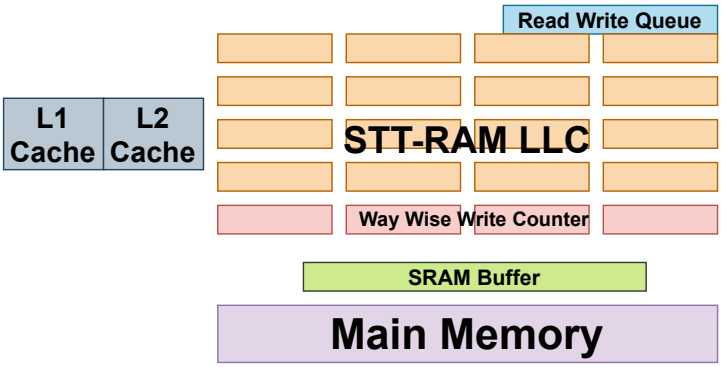

Figure 3.7: Proposed Architecture of LiveWay.

3.4.2 Working of LiveWay

The main idea is to more aggressively bypass selective writes from the heavily written ways. The LSC represents the reuse probability of a block in LLC, determined by observer sets in both caches and a table in L2; it increments when L2 frequently accesses the block within a time threshold, with a higher LSC indicating greater reuse potential, while a block starts with an LSC of 0.

Bypassing of writes are mainly dependent on two parameters: LSC and WWC. Figure 3.8 depicts how writes are bypassed based on the LSC and the WWC. At any moment if a way is having maximum write count, the way is treated as write-hot. Bypassing is done mainly from the top W write-hot ways. The value of W effectively determines the window size (WS). If a way is in the window and there are any incoming write backs to that way with LSC as 2 or 3, the write to that specific location is carried out. Any other incoming writes with LSC<2 are bypassed from the LLC. Out of these bypassed writes, the blocks having LSC as 1 are stored in the SRAM buffer and rest are written directly to the main memory.

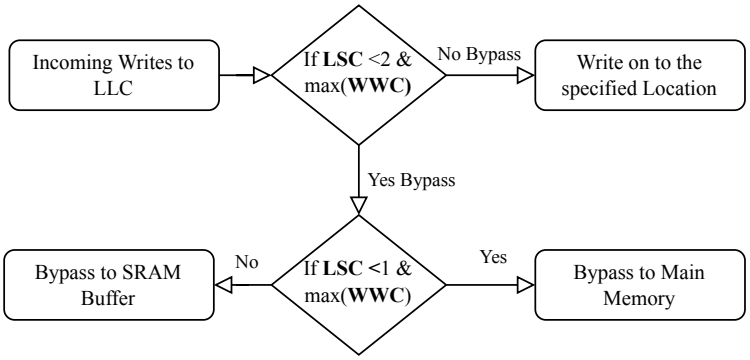

Figure 3.8: Flowchart of the Proposed Algorithm for LiveWay.

3.5 Experimental Setup

3.5.1 Simulator Setup

For all the experimental implementation of this work we use the ChampSim Simulator [97]. ChampSim replicates a diverse multi-core system featuring various cores and a custom memory hierarchy, allowing each out-of-order core to be individually configured according to specific requirements. Our experiments are conducted on a single and multi-core (mainly 4 core) system with three levels of cache hierarchy. L1 and L2 are SRAM based caches. The L3 (LLC) is designed with STT-RAM. For the multi-core setup L3 is a shared cache. The ChampSim is modified to support the STT-RAM LLC. Table 3.4 shows us the detailed parameters of the system used. WL denotes Write Latency while RL denotes Read Latency. All our timing parameters have been calculated with the help of CACTI [1].

Table 3.4: Simulation parameters of the baseline single-core system.

System Components	Parameters
Core	Out-of-order, bimodal branch predictor, 4 GHz with 6-issue width, 4-retire width, 352-entry ROB
L1I	32 KB, 8-way, 4 cycles
L1D	48 KB, 12-way, 5 cycles
L2	1MB, 8-way, 10 cycles, LRU
LLC	4MB, STT-RAM, 16-way, WL: 100 cycles, RL: 20 cycles, LRU
MSHRs	8/16/32 at L1I/L1D/L2, 64/core at the LLC
DRAM controller	64-entry RQ and WQ, reads prioritized over writes, Burst write: 6/8th of queue size
DRAM chip	4KB row-buffer per bank, open page, burst length 16, t_{RP}: 12.5ns, t_{RCD}: 12.5ns, t_{CAS}: 12.5ns

We compare our work with existing approaches namely, PoLF [12], WAD [15],SwS [12], G-OAS [15], SWWR [14] and DWAWR [14] and the baseline STT-RAM LLC. The baseline LLC uses LRU replacement policy and has no wear-leveling policy implemented. In PoLF the flush threshold (FT) value has been assigned as 16 for the simulations. WAD uses the clean-LRU block displacement algorithm along with a 3-bit saturating counters (SC). SwS only helps in swapping the mapping of two sets at once and all other sets are shifted gradually. Swapping only occurs if the total write count of the cache crosses a certain swap threshold (ST). G-OAS groups the cache sets into small groups (4,8,16,etc.) and swaps the write hot sets with the cold sets within the different groups. G-OAS(4) indicates that 4 groups of sets are considered based on their write hotness similar to our configuration. SWWR uses a window size of 4 ways with a 12-bit write counter per window. DWAWR uses a 11-bit write counter per way to count the writes. Different configurations of our work are shown in Table 3.5.

We implement PROLONG with varying configurations based on bypass aggressiveness

and SRAM buffer size. PRO(x%) indicates that the bypass aggressiveness of PROLONG is x%. Our work uses only an 8-bit counter per set for WSC because it is enough to calculate the writes over our chosen epoch size. An extra 2-bit counter is used for every set along with the WSC which is used to store the WH bucket value.
Similarly, varying buffer sizes are experimented with 3 different window sizes (no. of write-hot-ways): 1, 2 and 4 in case of LiveWay. Different LiveWay configurations are represented as xWSy; here x is the buffer size (in KBs) and y is the window size. The access latency of the SRAM buffer used in LiveWay is also considered in experiments.
For our experiments, we consider epoch size to be 10^5 instructions. In a single-core setup for each workload, the simulation commences with a 100 million instruction warm-up phase, followed by an additional 1 billion instruction run to reach completion. In case of multi-core setup a 50 million instruction warm-up and a 250M instruction run was deployed for our quad core systems.

Table 3.5: Different PROLONG configurations used.

Bypassing Aggressiveness %	**Buffer Sizes**
PRO(2%)	32KB, 64KB, 128KB, 256KB and 512KB
PRO(5%)	32KB, 64KB, 128KB, 256KB and 512KB
PRO(10%)	32KB, 64KB, 128KB, 256KB and 512KB
PRO(15%)	32KB, 64KB, 128KB, 256KB and 512KB

3.5.2 Workloads

We have used write intensive benchmarks from SPEC2006 benchmark suites [98]. We first simulated the SPEC2006 workloads and identified those workloads that are write intensive in LLC. A few write intensive graph benchmarks from the GAP benchmark suite[99] were also used for our simulations. Table 3.6 identifies all the write intensive workloads on which our single-core simulations have been run. We have also run our simulations on AI workloads like deepsjeng, leela, and exchange2 in order to capture ow they behave with our proposed techniques.

Table 3.6: Workloads for single-core system.

Benchmark Suite	**Write Intensive Workloads**
SPEC2006	cactusADM_734B, gcc_13B, GemsFDTD_109B, lbm_94B, leslie3d_94B, libquantum_964B, mcf_46B, milc_360B, soplex_66B, sphinx3_883B, wrf_1212B, xalancbmk_99B, zeusmp_100B
GAP	bc-12, bfs-10, cc-13, pr-5, sssp-5
AI Workloads	deepsjeng, leela, exchange2

For the quad-core setup, we use 4 different workload mixes. Each mix has different characteristics. The workload mixes are depicted in Table 3.7. 'L' denotes low write intensive workloads and 'H' denotes high write intensive workloads. 'G' denotes Graph workloads.

Table 3.7: Workloads for Quad-Core System.

Mix Type	Workloads
Mix1-LLHH	gobmk, gromacs, mcf, libquantum
Mix2-HHHH	mcf, libquantum, lbm, xalancbmk
Mix3-LLLL	gobmk, gromacs, gamess, namd
Mix4-GGGG	pr (page rank), bc (betweenness centrality), bfs (breadth first search), sssp (single source shortest path)

3.6 PartA: Results and Analysis

3.6.1 Single-core Analysis

In this section we compare different wear-leveling techniques with the baseline STT-RAM LLC considering various metrics. Among the different configurations of PROLONG, we consider the configurations only with 512KB buffer in this analysis with varying degrees of aggressiveness. A detailed sensitivity analysis with other PROLONG configurations is given in Section 3.6.3. Figure 3.9 shows the reduction in InterV of different wear-leveling techniques, normalised to baseline. It can be observed from the figure that, on an average the InterV reduces by around 82% in the case of PROLONG (15%) configuration. Among the existing techniques, SWS and G-OAS(4) achieve InterV reduction of 89% and 78%, respectively. Remaining techniques perform poorly. The main reason for the reduction in InterV in the case of PROLONG (10% & 15%) is aggressive bypassing of writes from the heavily written sets. However, it has been observed that excessive aggressive bypassing is not required as even the 2% aggressiveness achieves 74% reduction in InterV over the baseline. The higher the aggressiveness of bypass the greater the IPC of the system. More detailed analysis about aggressiveness is discussed in Section 3.6.3.

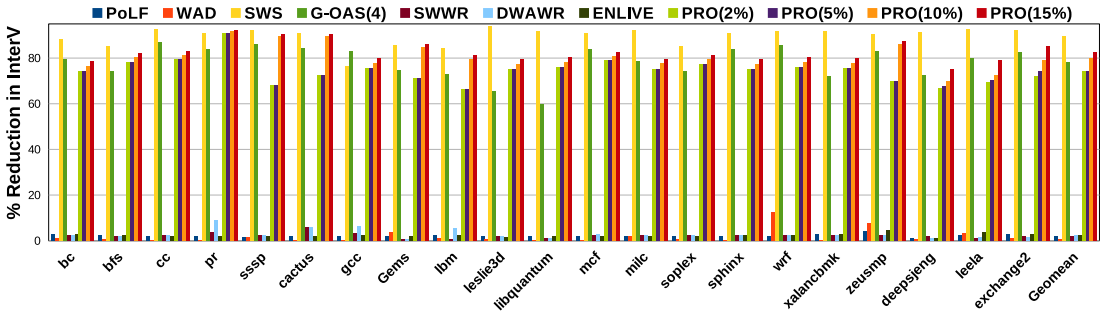

Figure 3.9: Percentage Reduction in InterV in Single-core System (higher the better).

The reduction in IntraV of PROLONG is comparable with the existing wear-leveling techniques. As shown in Figure 3.10, the reduction in IntraV for PROLONG is just above 20% normalized to baseline. This is comparable with previous state-of-the-art 16% for PoLF and 28% WAD which are specialized IntraV reduction techniques. The reason behind such reduction in the existing techniques is that all these techniques try to distribute the writes among the different ways of the set. There are two main reasons for

reducing the IntraV by PROLONG. First, it has been observed that the IntraV is high on the sets having more writes. Hence, bypassing writes from heavily written sets also helps in reducing write variation within the set. Second reason is based on an insight from PoLF [12]. The idea discussed in PoLF achieves significant improvement in IntraV by just invalidating random cache blocks. Bypassing writes also means invalidating the existing block that belongs to an heavily written set of the LLC. Hence, combining these two important reasons, PROLONG also shows competitive reduction (around 20%) in IntraV.

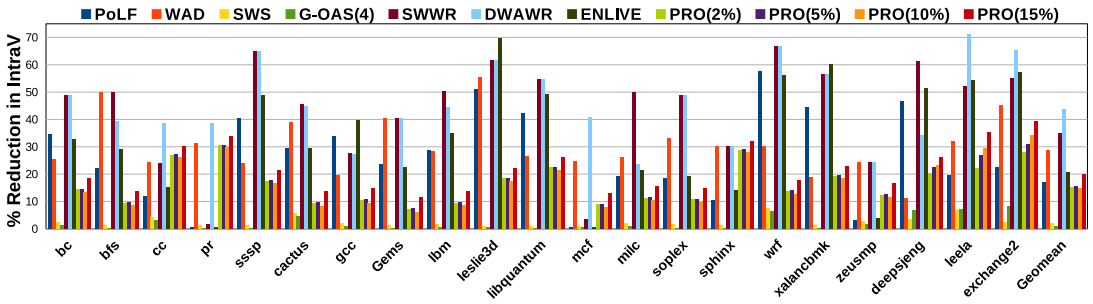

Figure 3.10: Percentage Reduction in IntraV in Single-core System (higher the better).

The lifetime of a STT-RAM based LLC is dependent on the maximum write count of a single cache line. Bypassing writes from heavily written sets not only reduces the total number writes performed in the STT-RAM LLC but also reducing the maximum write count. Hence, PROLONG performs less number of writes in the LLC as compared to the other techniques. The existing techniques sometimes even increase the number of writes because of invalidating important blocks from the LLC while remapping. These important blocks, the blocks that are likely to be reused in the future, need to be fetched from the main memory again, leading to more writes at the LLC. Missed important blocks can potentially lead to performance degradation. However, in PROLONG, the write bypassing takes place on the basis of a liveness score of the block. The liveness score indicates the possibility that the block can be reused again in the near future. Therefore, the possibility of bypassing an important block is less as compared to the other state-of-the-art.

Figure 3.11 shows the comparison in the number of writes by different techniques as compared to the baseline, i.e. Normalized Total Writes. It can be observed from the figure that the reduction in writes is more in PROLONG whereas in other techniques that do not employ write bypassing and the writes are either more or equal to that of the baseline (LRU). Figure 3.11 depicts the various configurations of PROLONG with varying degrees of aggressiveness (2-15%).

Figure 3.12 shows the improvements in lifetime over the baseline by the various wear-leveling techniques considered. As mentioned in Chapter 2, the lifetime depends on write variations and the maximum number of writes in a cache line. PROLONG improves all the three parameters as shown in Figure 3.9, 3.10, and 3.11. Therefore, Figure 3.12 shows significantly better lifetime than the other techniques. All the other existing techniques can enhance the lifetime at maximum by 9× wheres PROLONG enhances it

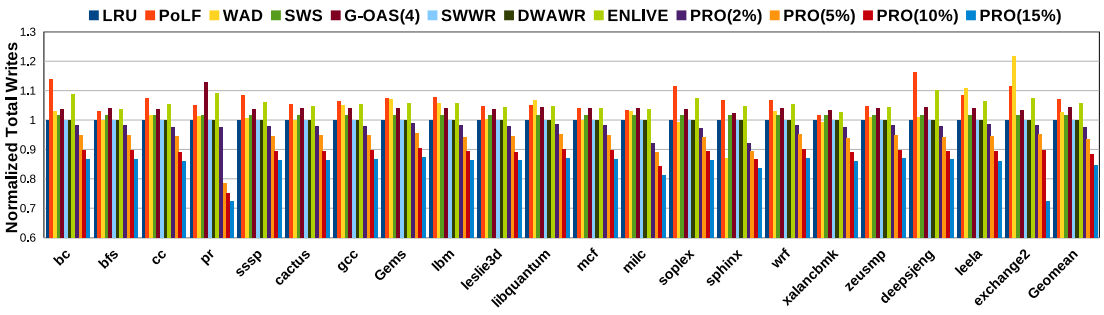

Figure 3.11: Comparison of Normalized Total Write Count in single-core Systems (lower is better).

by nearly 23×, for the single-core configuration. Here, the lifetime is calculated as the normalized lifetime over baseline.

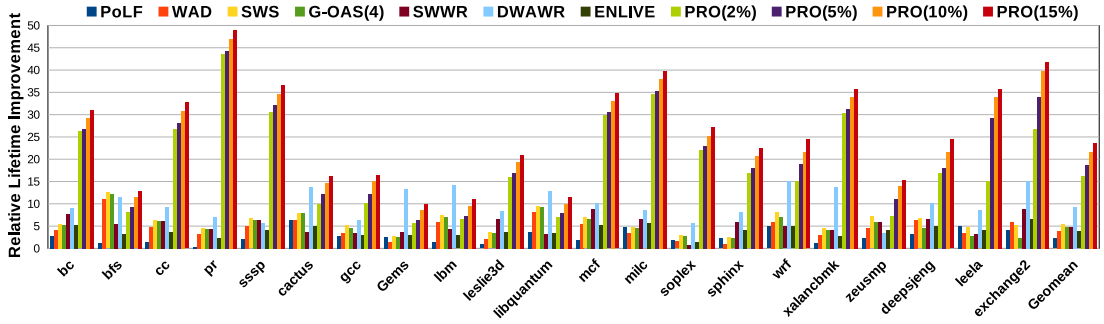

Figure 3.12: Comparison of Relative Lifetime Improvement in single-core Systems (higher the better).

Impact on performance: While achieving significant improvement in lifetime, PROLONG does not degrade (on an average) the performance of the system. Normalized IPC is used as the metric to show the various performance improvements as compared to the other state-of-the-art. Figure 3.13 shows that PROLONG matches the performance of baseline and in some cases namely, bc, cc, pr, sssp, gcc and sphinx, PROLONG achieves more than 10% performance improvement over the baseline. The average improvement ranges from 0-2%. The reason behind the improvement is because of bypassing the less important blocks from the cache. As discussed in Density [13], bypassing less important blocks reduces the congestion in the read/write queue at the LLC and hence improves the efficiency of the LLC. Because of lesser aggressive bypassing than discussed in Density [13], PROLONG does not attain performance improvements similar to Density. The other existing wear-leveling techniques, PoLF, WAD, SWS and G-OAS(4) show a significant drop in performance. Similar to PROLONG, SWWR and DWAWR almost achieve baseline performance. This is the main benefit of PROLONG as it improves the lifetime significantly as compared to the existing techniques without degrading the performance.

Improvement analysis: Figure 3.14(a) shows the geomean reduction in terms of write variations both InterV and IntraV as compared to the baseline and other state-of-the-art configurations. Figure 3.14(b) shows us the reduction in the total write count of

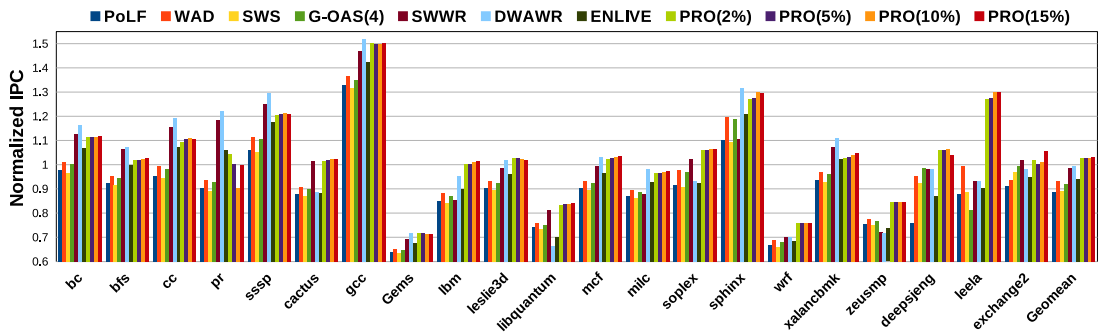

Figure 3.13: Comparison of Normalized IPC in single-core Systems (higher the better).

PROLONG that occurs due to write bypassing. It is depicted as the percentage reduction in write count as compared to baseline. Other wear leveling techniques show an increase in the total write count as compared to baseline, i.e. a negative percentage. The number of writes in these state-of-the-art wear leveling techniques may increase but we see that the maximum write count in a cache block does not increase due to effective wear leveling. Figure 3.14(c) shows us how the lifetime is effected effected by the various existing techniques considered along with PROLONG. Figure 3.14(d) deals with the effect of the state-of-the-art on the performance calculated as Normalized IPC, over baseline. The improvement shown in these figures are the geometric mean values of all the workloads considered for study. It can be observed that PROLONG reduces both InterV and IntraV by almost 84% and 19% respectively. The write count in existing techniques is more than the baseline wheres in PROLONG it reduces because of its efficient bypassing. As the bypass aggressiveness of PROLONG increases, the reduction in write count increases due to the bypassing of unimportant blocks. The significant improvement in write variations reduction and write count reduction makes PROLONG improve the lifetime as shown in Figure 3.14(d). An important point to observe here is that the higher the bypass aggressiveness, the higher the lifetime.

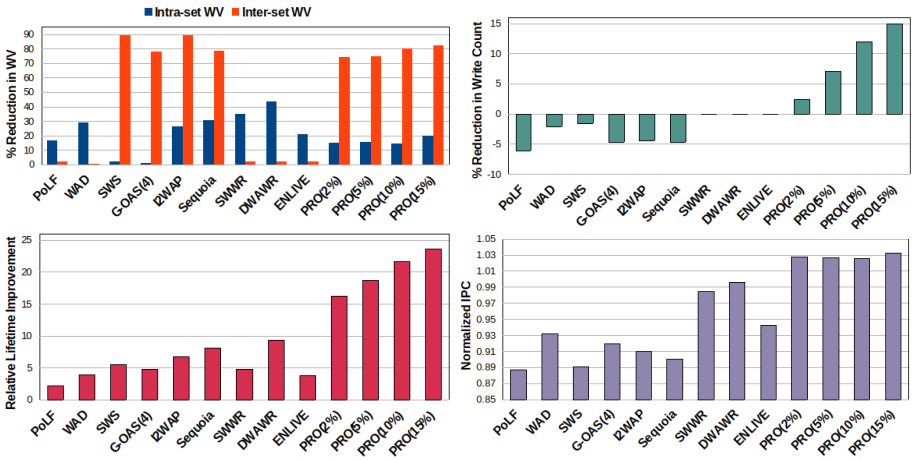

Figure 3.14: Average Comparison of PROLONG in Single-core Systems with state-of-the-art.

3.6.2 Multi-core Analysis

Multi-core experiments show the behavior similar to single-core experiments but in an amplified scale. Figure 3.15 and 3.16 show the comparison of different wear-leveling techniques for reduction in InterV and IntraV. The reduction of IntraV by PROLONG is slightly less than single-core i.e. a maximum of 12%, while the InterV reduction is greater than the single-core system nearly by 89%. We have also conducted experiments with various workloads, including mcf, libquantum, and xalancbmk, in multi-core environments where a single workload is distributed across all system cores. Our observations indicate that the results closely align with those seen in scenarios where multiple write-intensive benchmarks are executed on separate cores.

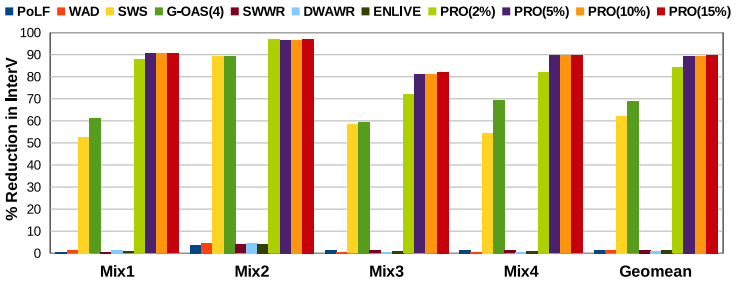

Figure 3.15: Percentage Reduction in InterV for Multicore Systems (higher the better).

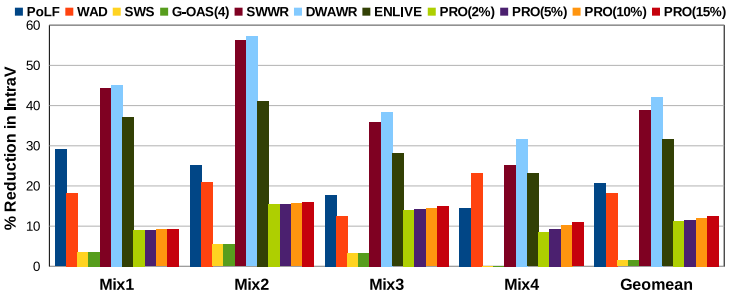

Figure 3.16: Percentage Reduction in IntraV for Multi-core Systems (higher the better).

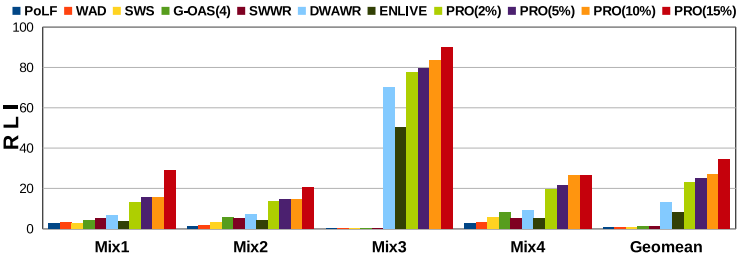

Figure 3.17: Relative Lifetime Improvement (RLI) for Multi-core Systems (higher the better).

For multi-core setup, Figure 3.17 shows that PROLONG enhances the lifetime up to 35×, a close observation shows that the enhancement is gradual and similar to single-core for *Mix2*. However, for *Mix1*, *Mix3* and *Mix4* PROLONG shows a much higher enhancement.

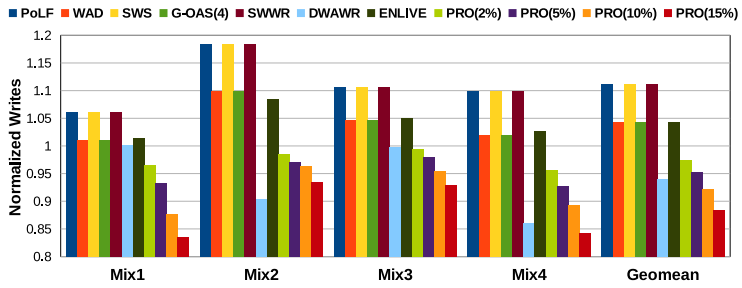

Figure 3.18: Normalized Total Writes w.r.t. baseline (LRU) for Multi-core Systems (lower is better).

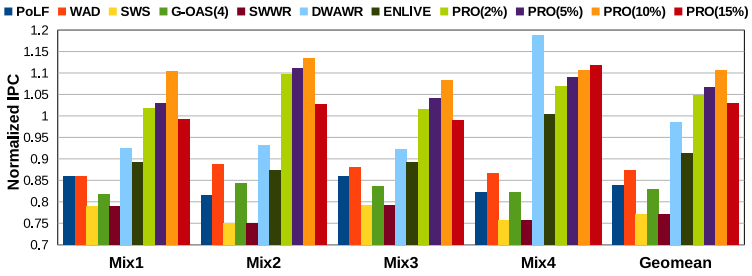

Figure 3.19: Normalized IPC w.r.t. baseline (LRU) for Multi-core Systems (higher the better).

This occurs because of extensive bypassing of writes that will not be accessed again in the near future as these workloads have a major mix of read intensive instructions and graph workloads. The write count in the LLC decreases drastically with increasing aggressiveness of bypassing as depicted in Figure 3.18. Figure 3.19 shows the performance (Normalized IPC) improvements over the baseline. It can be observed from the figure that the performance of PROLONG is almost similar as baseline. However, the performance of the existing wear leveling techniques goes below the baseline.

3.6.3 Sensitivity Analysis

In this section we discuss about the various possible configurations of PROLONG and reason the optimum configuration. For the analysis of this section, PROLONG is considered with five different sizes of SRAM buffer: 32KB, 64KB, 128KB, 256KB, and 512KB. For all the earlier experiments we considered buffer size of 512KB. Each buffer is divided into multiple parts of 32 entries just like a set-associative cache such that the access time of the buffer can be reduced. Considering the latencies in Table 3.8, we perform experiments in this section. For each buffer size, PROLONG considered four categories of bypass aggressiveness: 2%, 5%, 10%, and 15%.

Figure 3.20 shows the improvement in PROLONG over baseline in a single-core system. The improvements are shown for write variation, write count, lifetime and performance. It can be observed from Figure 3.20(a) that the improvements in IntraVs are almost similar with slight increments in reduction with increment in buffer size. The InterVs slightly varies over different PROLONG configurations. However, the variations are within 6%.

Table 3.8: The important hardware parameters of the SRAM buffer used in PROLONG. The parameters are extracted from CACTI [1].

Buffer Size	**Access Latency**	**Static Power**	**Dynamic Energy per Access**	**Area Consumption**
32KB	4.3ns	1.78nJ	19.44mW	6.1mm^2
64KB	4.4ns	1.81nJ	33.16mW	6.55mm^2
128KB	4.5ns	1.83nJ	60.74mW	7.4mm^2
256KB	4.8ns	1.86nJ	110.21mW	8.9mm^2
512KB	5.3ns	1.91nJ	168.65mW	10.45mm^2

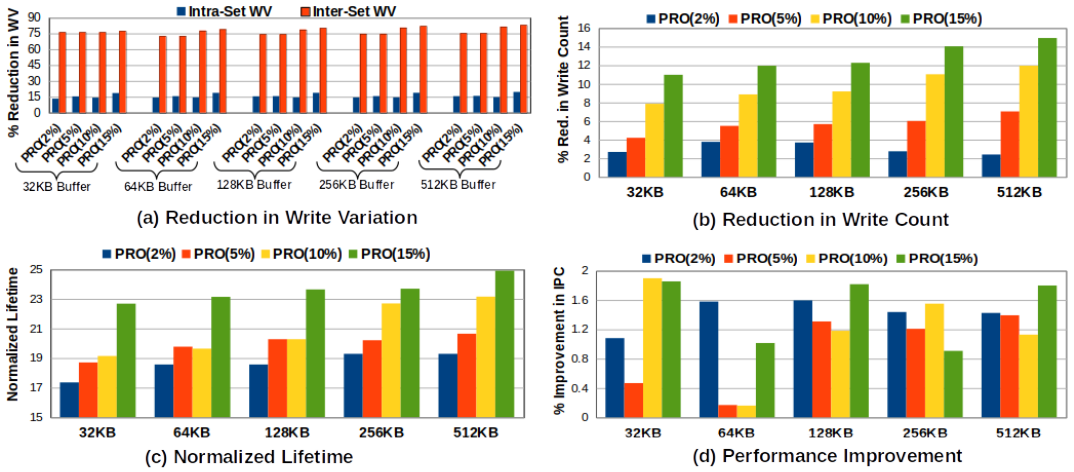

Figure 3.20: Comparison of different PROLONG configurations with baseline in a single-core environment. PRO(x%) means the bypass aggressiveness of PROLONG is x%. Each bypass aggressiveness value is experimented with 32KB, 64KB, 128KB, 256KB, and 512KB of buffer size.

Figure 3.20(b) shows that the number of writes in LLC reduces with increase in bypass aggressiveness. A 512KB buffer with 15% bypass aggressiveness depicts a 14% decrease in the overall write count. The change in normalized lifetime (as shown in Figure 3.20(c)) over the different PROLONG configurations varies up to 1.5×. The minimal lifetime improvement is only 17× while the maximum is nearly 25×. Figure 3.20(d) shows that there is no significant improvement in the performance with an increase in the SRAM buffer size. The main reason behind this behaviour is the high write latency of the larger SRAM buffers. The write latency of the SRAM buffer increase slightly with its increasing size as depicted in Table 3.8.

Hence, from the above discussion it can be concluded that a PROLONG configuration with large SRAM buffer is not improving noticeably. Furthermore, they have a larger hardware overhead that make it an unattractive prospect. Buffer sizes that are small are also not optimal as it lags behind in the lifetime improvement. Therefore, buffer sizes that are in the middle like 128KB can provide the best of both worlds with minimal hardware overhead and high normalized lifetime for highly aggressive write bypassing is

more advisable. The results from multicore setup also reflects the similar insights but in a more heightened sense.

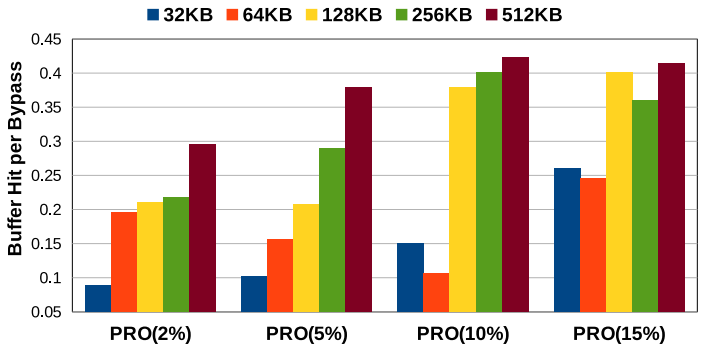

Figure 3.21: Buffer hit per write bypass in different configurations of PROLONG (higher the better).

3.6.3.1 Importance of SRAM buffer

In this section we discuss the importance of using an SRAM buffer in PROLONG. Since the bypass aggressiveness that we used for PROLONG is more than the aggressiveness used in Density [13], there is a likelihood that PROLONG bypasses important blocks. The buffer is used to store such blocks. Figure 3.21 shows the buffer hit per LLC bypass. It can be observed from the figure that the hit rate in buffer increases with increasing buffer size. This implies that the larger the buffer it stores more important blocks with high probability of being reused again. As the bypass aggressiveness increases, more loads are coming to the SRAM buffer and hence the hit rate is also increasing. However, from the figure, it can be observed that without the SRAM buffer there will be multiple request which need to be served from main memory. Hence, the SRAM buffer plays an important role from the performance perspective.

Effect of AI workloads on the SRAM BUffer: With the higher reuse rate of memory blocks in the AI workloads (deepsjeng, leela, and exchange2), there is a higher buffer hit rate per bypass. Figure 3.22 shows the buffer hit per write bypass for AI workloads with increasing bypass rate and increasing buffer size. Therefore, it is evident that there is a need for an SRAM specifically for AI workloads.

3.6.3.2 Comparison with other write bypassing techniques

Write bypassing was initially employed to reduce the number of writes and improve the performance of STT-RAM LLCs, as described in [13]. In contrast, PROLONG leverages this technique to extend the lifetime of the LLC by minimizing both write variations and the overall number of writes. Figure 3.23 illustrates the relationship between PROLONG and the Density technique [13]. From the figure, it is evident that in single-core systems, PROLONG achieves a significant lifetime improvement of 22×, whereas Density offers only a negligible enhancement compared to the baseline. This disparity arises because

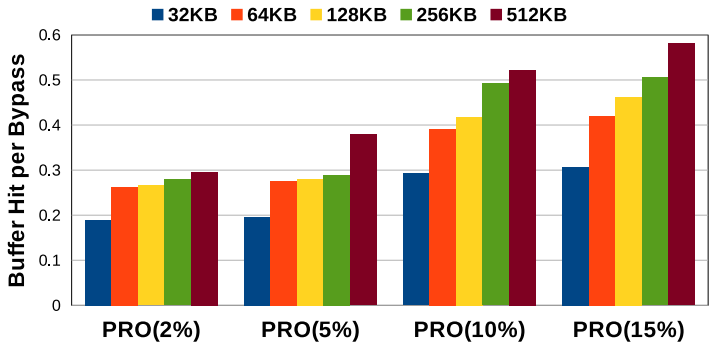

Figure 3.22: Buffer hit per write bypass in different configurations of PROLONG for AI workloads (higher the better).

Density primarily focuses on bypassing writes to alleviate congestion in the read/write queue, bypassing less critical blocks across the entire LLC. PROLONG, on the other hand, performs more aggressive bypassing, but only for selected sets. Despite this, Density delivers greater performance gains over the baseline than PROLONG, as shown in Figure 3.24, highlighting its focus on performance rather than longevity.

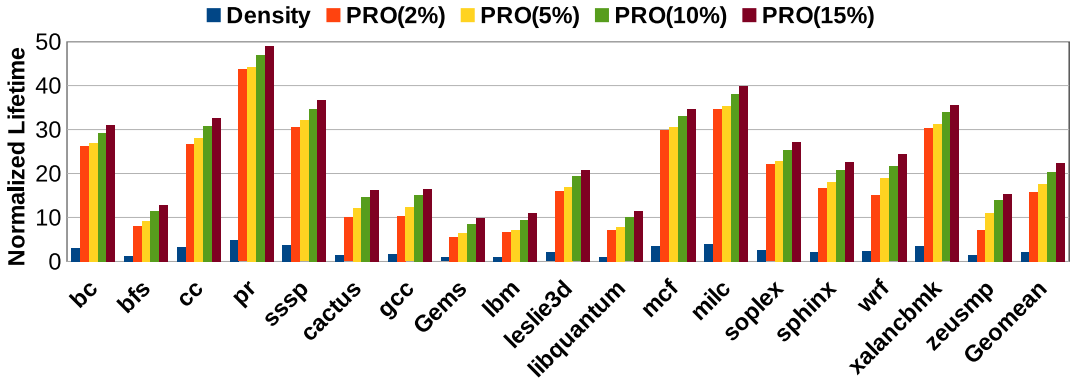

Figure 3.23: Comparison of Normalized Lifetime of PROLONG with Density (higher the better).

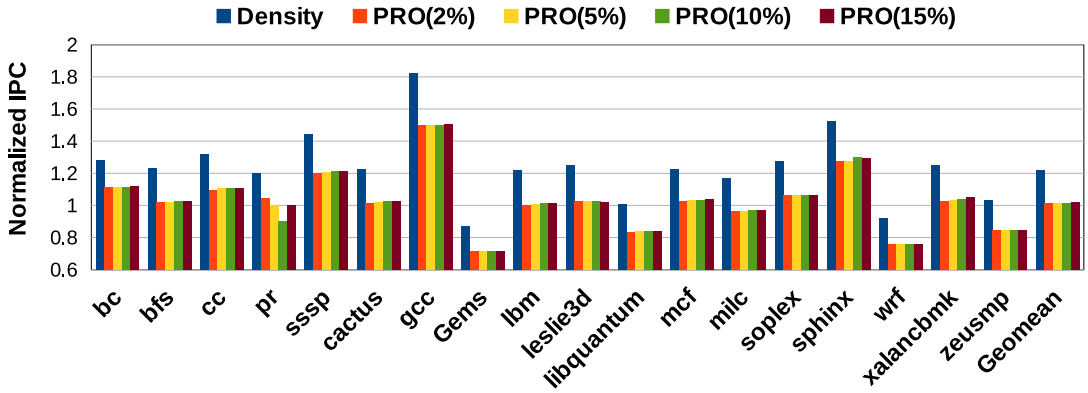

Figure 3.24: Comparison of Normalized IPC of PROLONG with Density (higher the better).

3.6.3.3 Lifetime Comparison Analysis

Table 3.9 presents a comparison of the normalized lifetimes achieved by various state-of-the-art techniques, juxtaposed with different configurations of the PROLONG method, tested across a range of write-intensive workloads. These workloads include benchmarks from the SPEC2006 suite and highly write-intensive graph workloads from the GAP benchmark suite. The lifetime values are normalized against a baseline configuration, where the STT-RAM based LLC employs an LRU replacement policy without any wear-leveling strategies. For all PROLONG configurations, a 512KB SRAM buffer is used. The results indicate that the graph benchmarks generally exhibit a more significant improvement in normalized lifetime, reaching up to nearly 49× for the PRO(15%) configuration. In comparison, the highest normalized lifetime for the SPEC2006 workloads reaches 40× for the same PRO(15%) configuration. Table 3.9 also shows that the geometric mean normalized lifetime for state-of-the-art techniques reaches a maximum of 9× in the case of DWAWR. However, the PRO(15%) configuration demonstrates the highest geometric mean normalized lifetime, reaching 22.32×.

A clear trend emerges from Table 3.9, showing a steady increase in normalized lifetime as the degree of bypass aggressiveness in the PROLONG configurations increases, highlighting the efficiency of PROLONG in enhancing the endurance of STT-RAM-based LLCs under heavy write workloads.

Table 3.9: Normalized Lifetime of Write Intensive Workloads.

Workloads	PoLF	WAD	SWS	G-OAS(4)	SWWR	DWAWR	ENLIVE	Pro(2%)	Pro(5%)	Pro(10%)	Pro(15%)
GAP											
bc	2.7548	4.0446	5.5163	5.2186	7.6479	9.0643	5.2014	26.2267	26.8707	29.2894	31.1072
bfs	1.2195	11.0368	12.5424	12.1622	5.3968	11.4094	3.3082	8.1433	9.2105	11.4094	12.9252
cc	1.3889	4.8851	6.4140	6.1047	6.1047	9.2814	3.7468	26.7361	28.0702	30.8244	32.7273
pr	0.2299	3.2564	4.6191	4.3062	4.4311	7.1253	2.3305	43.6573	44.2514	47.0489	48.9325
sssp	2.0408	5.1051	6.7073	6.3830	6.3830	5.7402	4.2119	30.5970	32.0755	34.6154	36.7188
SPEC2006											
cactus	6.4516	6.4516	7.8431	7.8431	3.7736	13.7931	5.1126	10.0000	12.2449	14.5833	16.1972
gcc	2.8090	3.3898	5.1724	4.5714	3.3898	6.3953	3.0994	10.2409	12.2699	15.0943	16.5605
Gems	2.6066	1.4052	2.8504	2.6066	3.5885	13.3508	3.0976	5.6098	6.3882	8.5213	9.8985
lbm	1.5209	5.9524	7.4447	7.0140	4.2969	14.1026	2.9089	6.5868	7.2289	9.4262	11.0187
leslie3d	0.8715	2.2075	3.5794	3.3482	6.6820	8.4309	3.7767	16.0401	16.9192	19.3299	20.8877
libquantum	3.6232	8.1285	9.5785	9.3690	3.2491	12.8205	3.4361	7.1161	7.9245	10.0000	11.5010
mcf	1.8569	5.4843	6.9314	6.5468	8.8170	10.1115	5.3370	29.9123	30.4846	33.0638	34.7589
milc	4.8128	3.4301	4.8128	4.5333	6.5217	8.4871	5.6673	34.7079	35.3280	38.0282	39.8335
soplex	1.9608	1.5625	2.9703	2.7668	0.7752	5.6911	1.3680	22.0657	22.9314	25.3012	27.1394
sphinx	2.2364	0.9464	2.5641	2.2364	5.9603	8.1081	4.0983	16.7883	18.0812	20.7547	22.6054
wrf	4.9020	5.9406	8.0808	7.0000	4.9020	15.0538	4.9020	15.0538	18.8889	21.5909	24.4186
xalancbmk	1.2500	3.0534	4.5161	4.1131	4.1131	13.7640	2.6816	30.4348	31.2804	33.8843	35.6784
zeusmp	2.2727	4.6512	7.1429	5.8824	5.8824	3.4483	4.0775	7.1429	11.1111	13.9241	15.3846
deepsjeng	3.1946	6.4516	6.7073	4.5333	6.6820	10.1115	4.9383	16.7883	18.0812	21.5909	24.4186
leela	5.1128	3.3898	4.8431	2.7668	3.2491	8.4871	4.1809	15.0538	29.2894	33.8843	35.6784
exchange2	4.1897	5.9524	5.1724	2.2364	8.8170	15.0538	6.5034	26.7361	33.8843	39.8335	41.8335
Geomean All	2.2187	3.9639	5.5650	4.7869	4.7784	9.3745	3.7918	16.2848	18.6485	21.6644	23.6191

We can clearly observe that there are some anomalies in the case of DWAWR vs. PROLONG where some workloads like bfs, cactus, Gems, lbm, and zeusmp show some amount of higher normalized lifetime improvement in the case of DWAWR over some configurations of PROLONG, specifically with 2% bypass aggressiveness. This is mainly because only a small percentage of bypass will not actively wear level writes from the high write intensive regions in the LLC, an increase in the bypass aggressiveness shows that the lifetime improves gradually and is eventually equivalent or better than the lifetime of DWAWR.

3.6.3.4 SPEC2017 vs SPEC2006 vs GAP

This section provides a detailed comparison of the results from various benchmarks, including SPEC2017, SPEC2006, and GAP. All the data presented here are derived from single-core systems running write-intensive workloads, with the geometric mean (geomean) values used for representation in the accompanying graphs. For this analysis, the SRAM buffer size has been primarily set to 512KB.

Figure 3.25 illustrates the subtle variations in normalized IPC across different workloads. The graph reveals that the trends between SPEC2017 and SPEC2006 benchmarks are quite similar. However, the GAP benchmarks, which involve a more intensive write workload, show improved normalized IPC values in comparison to their counterparts. The maximum geomean normalized IPC recorded was 1.025 for both SPEC2017 and SPEC2006 benchmarks, while GAP benchmarks achieved a slightly higher value of 1.12, demonstrating the benefit of our proposed techniques in write-heavy scenarios.

Figure 3.26 offers a comprehensive comparison of the percentage reduction in InterV across different workloads implemented on the proposed architectures. Both SPEC2006 and SPEC2017 exhibit similar trends, but the differences between the various configurations in the SPEC2017 benchmarks are more pronounced. The results for GAP benchmarks show a more significant reduction in InterV, driven by the higher intensity of write operations associated with these benchmarks.

Figure 3.27 highlights the percentage reduction in IntraV across the three benchmark sets. The data reveal that SPEC2017, SPEC2006, and GAP benchmarks all follow similar trends in terms of the reduction in IntraV. However, the GAP benchmarks show a more pronounced reduction compared to the other benchmarks. Furthermore, SPEC2006 demonstrates a greater reduction in IntraV than SPEC2017, which underscores the efficiency of the proposed techniques in managing intra-cache variations.

Finally, Figure 3.28 displays the Relative Lifetime Improvement (RLI) of various configurations compared to the baseline LRU configuration. Both SPEC2006 and SPEC2017 show similar trends and values, while the RLI for GAP benchmarks is notably higher. This confirms that our techniques offer significant benefits in terms of memory lifetime, particularly when handling highly write-intensive workloads.

In conclusion, we can assert that the proposed techniques perform more effectively on write-intensive GAP benchmarks compared to the SPEC benchmarks, especially in reducing cache-related inefficiencies and improving system longevity under heavy write loads.

3.6.4 Hardware Overhead and Energy Consumption

Table 3.10 presents the total area overhead of PROLONG for a given buffer size. Overhead comprises of write counters, WH bucket counters, LSC apart from the buffer. Since the other hardware components in PROLONG are almost the same as in the baseline, the additional energy consumed by PROLONG (AE_{pro}) can be calculated as $AE_{pro} =$

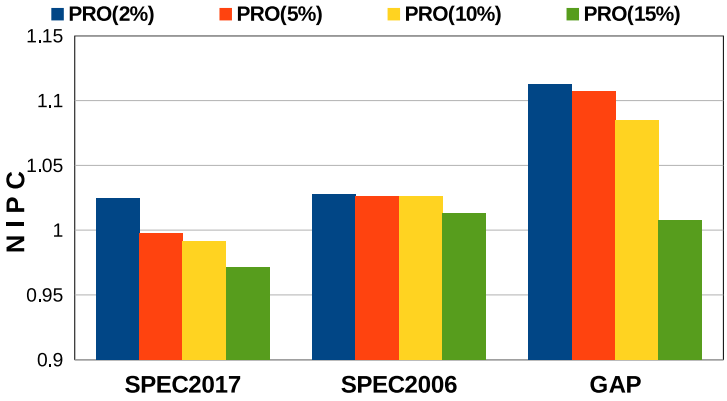

Figure 3.25: Normalized IPC of various different benchmarks run on various configurations.

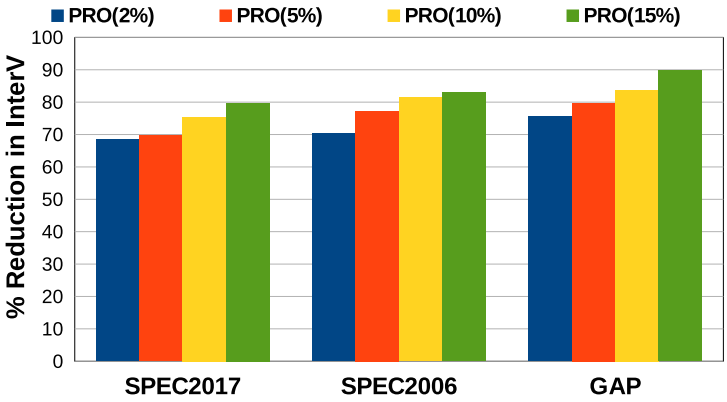

Figure 3.26: Reduction in InterV of various different benchmarks run on various configurations.

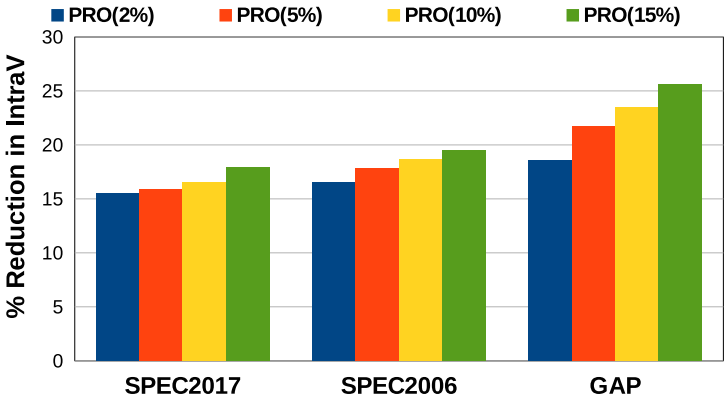

Figure 3.27: Reduction in IntraV of various different benchmarks run on various configurations.

$E_{buff} - E_{write}$. Where E_{buff} is the energy consumed by the SRAM buffer and E_{write} is the energy saved by reducing the number of writes. We have used CACTI [1] to model the SRAM buffer. Experimental analysis with all sizes of SRAM buffer found that the value of AE_{pro} is always negative, implying that PROLONG does not have any additional energy overhead with any buffer size, which is enough to get the benefit we are expecting.

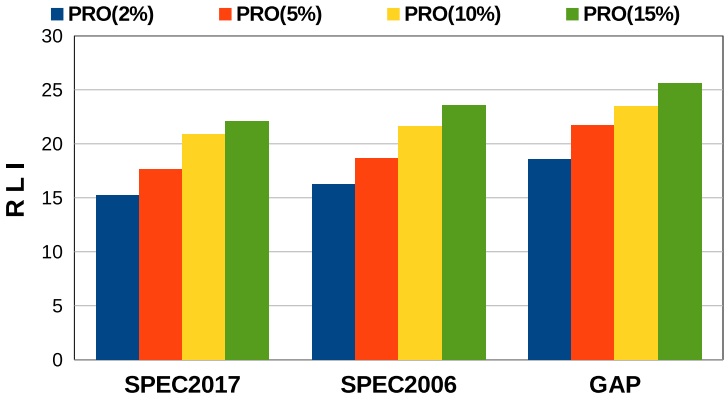

Figure 3.28: RLI of various different benchmarks run on various configurations.

Table 3.10: Storage Overhead of Prolong.

Buffer Size	Total Overhead	Overhead over Baseline
32KB	53KB	1.29%
64KB	85KB	2.08%
128KB	149KB	3.63%
256KB	277KB	6.76%
512KB	533KB	13.01%

3.7 PartB: Results and Analysis

3.7.1 Single-core Analysis

Figure 3.29 shows the percentage reduction in IntaV for different state-of-the-art wear-leveling techniques that aim to reduce the IntraV like PoLF[12], WAD[15], SWWR[14], DWAWR[14], and ENLIVE[48] as well as for the best configurations of LiveWay, i.e. LiveWay (128WS2) and LiveWay (64WS2). LiveWay (128WS2) shows a 53% reduction in IntraV while the maximum reduction shown by any other existing technique is 43% in case of DWAWR [14]. Figure 3.30 indicates that LiveWay (128WS2) has on geomean 21.23× improvement while the next best state-of-the art technique DWAWR shows only 9.37× improvement in RLI.

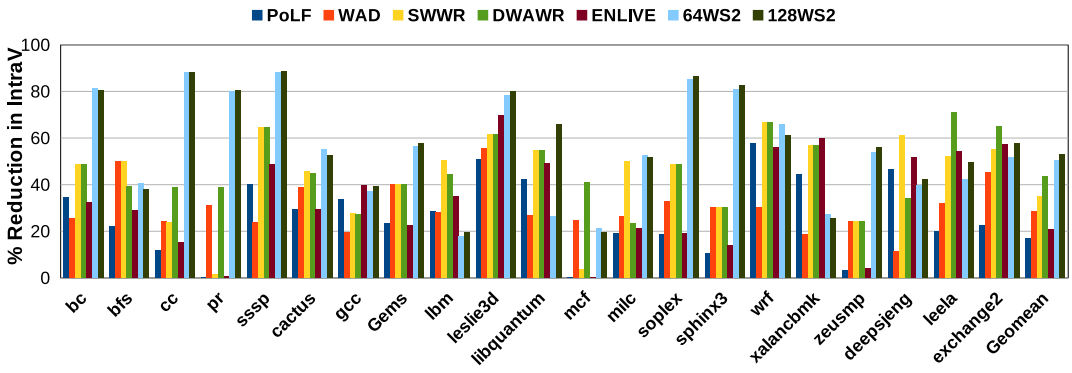

Figure 3.29: % Reduction in IntraV for Single-core Systems (Higher the better).

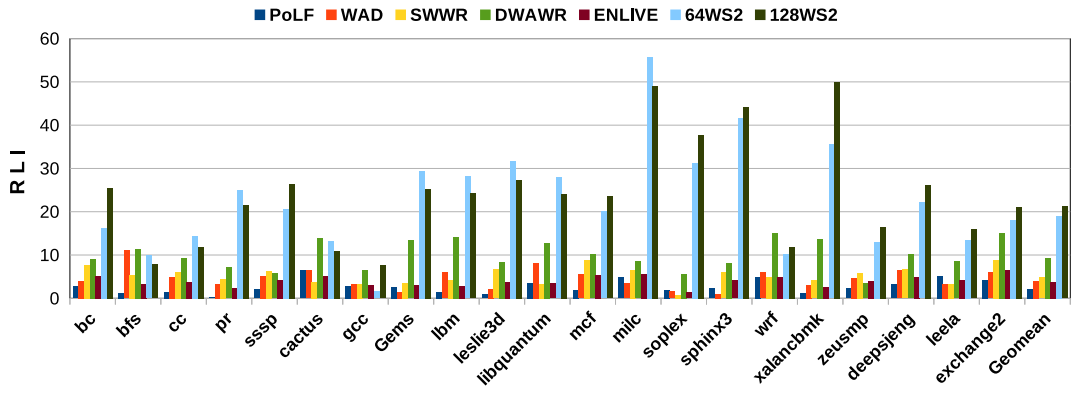

Figure 3.30: Relative Lifetime Improvement (RLI) (in times) for Single-core Systems (Higher the better).

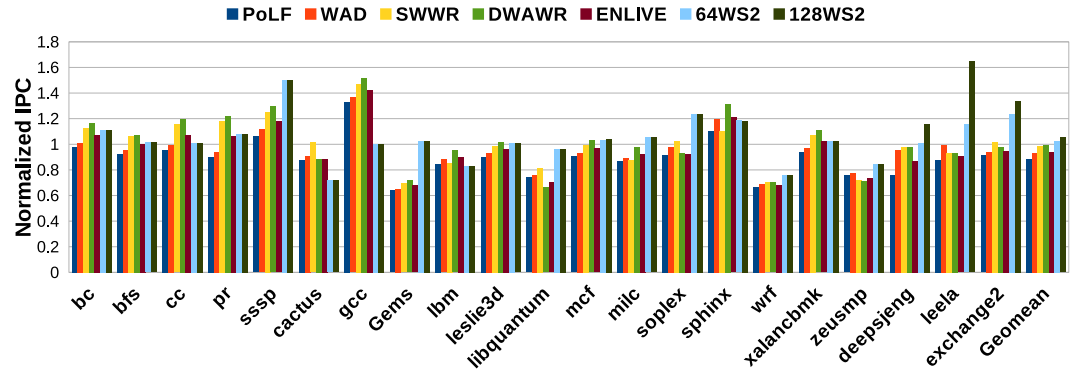

Figure 3.31: Geomean Normalized IPC for Single-core Systems (Higher the better).

Figure 3.31 depicts that LiveWay shows very little performance degradation as compared to other techniques. This is due to the benefits brought about by the decongestion in the read/write queue of LLC through write bypassing. The benefits in terms of IPC are not as much as Density [13] which has a higher bypass aggressiveness, because even if there is decongestion, bypassing writes will result in cache miss, and then the data needs to be fetched from the SRAM buffer or the MM, this will incur some extra latency.

3.7.2 Multi-core Analysis

The results for multi-core systems reiterate our findings from the single-core analysis but in a muffled manner. The resultant values are significantly lower than single-core values due to the extensive distribution of writes in a multi-core system having larger shared memory. Mix1 is a combination of both read and write-intensive workloads while Mix2 is a combination of write-intensive workloads and Mix3 is a combination of graph workloads only. Figure 3.32 depicts the impact of multi-core systems on the IntraV. Our proposed architecture brings about a maximum of 67% IntraV reduction for Mix2. Mix2 is write-intensive which is why the reduction in IntraV is the highest.

The lifetime of a STT-RAM based LLC is dependent on the maximum write count of a single cache line. Bypassing writes from heavily written sets not only reduces the total

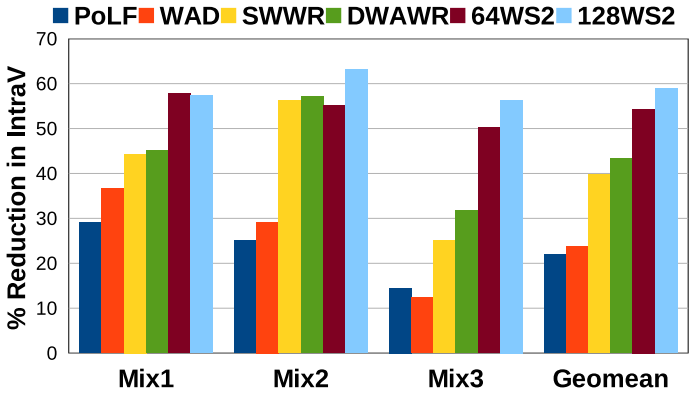

Figure 3.32: % Reduction in IntraV for Multi-core Systems.

number of writes performed in the STT-RAM LLC but also reduces the maximum write count. Figure 3.33 shows that in multi-core systems, LiveWay can bring about drastic lifetime improvement through average write reduction with the help of write bypassing. The average write count per block reduces drastically for LiveWay and therefore can directly affect the relative lifetime of the LLC. The RLI of LiveWay(64WS2) specially in case of Mix3 goes upto 38×, while in the case of LiveWay(128WS2) goes upto 45×.

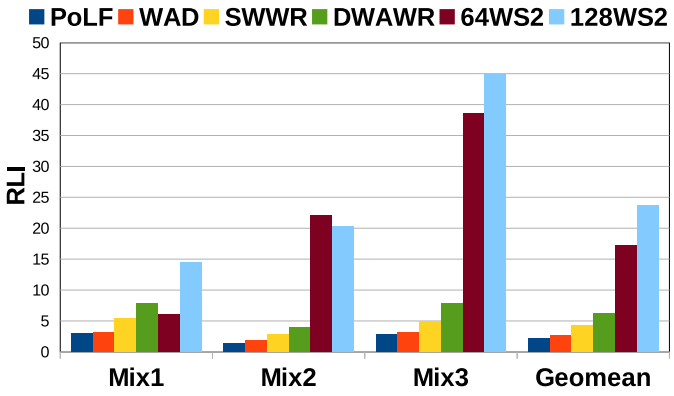

Figure 3.33: Relative LI (in times) for Multi-core Systems.

Also, the performance degradation in LiveWay is lower than that of the other techniques, as shown in Figure 3.34. The SRAM buffer used to store the selectively bypassed blocks causes 25% hits and, therefore, helps to increase performance as compared to the other state-of-the-art techniques.

3.7.3 SPEC2017 vs SPEC2006 vs GAP

This section provides a comparative analysis of benchmark outputs from SPEC2017, SPEC2006, and GAP, highlighting the variations in system performance across different workloads. All results presented are based on single-core systems executing write-intensive workloads, with the geometric mean (geomean) of the results used for graphical representation.

Figure 3.35 illustrates the subtle variations in normalized IPC across different workloads.

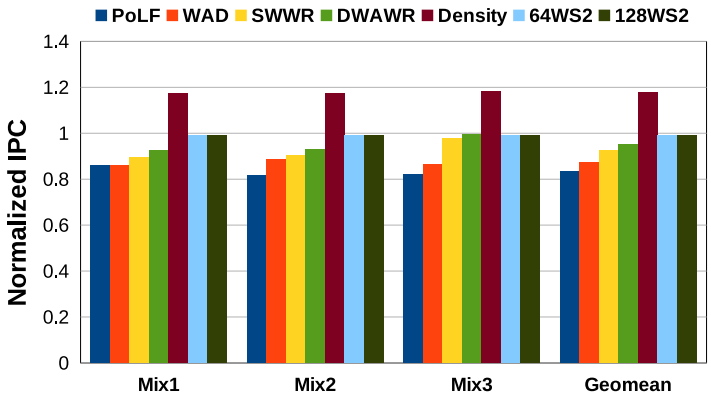

Figure 3.34: Normalized IPC for Multi-core Systems.

The observed trends indicate a strong similarity between SPEC2017 and SPEC2006 benchmarks. However, the GAP benchmarks, which are inherently more write-intensive, exhibit higher normalized IPC values compared to their SPEC counterparts. The maximum geomean normalized IPC values were found to be 1.15 for SPEC2017, 1.189 for SPEC2006, and 1.35 for GAP benchmarks, showing a consistent increase with larger window sizes. In this analysis, the SRAM buffer size was primarily set at 128KB.

Figure 3.36 provides an in-depth comparison of the percentage reduction in IntraV across different workloads when executed on the proposed architectures. While SPEC2006 and SPEC2017 follow similar trends, the variations in SPEC2017 benchmarks are more pronounced across different configurations. The GAP benchmarks, due to their high-intensity write operations, show significantly greater reductions in IntraV than SPEC benchmarks, reinforcing their distinct behavior under write-heavy conditions.

Figure 3.37 presents the Relative Lifetime Improvement (RLI) across various configurations in relation to the baseline LRU configuration. Once again, SPEC2006 and SPEC2017 display closely matching trends and values, whereas the GAP benchmarks show a markedly higher RLI, emphasizing the impact of intensive write operations on memory endurance.

From these observations, we conclude that our proposed techniques perform more effectively on the highly write-intensive GAP benchmarks compared to SPEC benchmarks. The heightened impact of our optimizations on GAP workloads suggests that our approach is particularly well-suited for environments with intensive memory write operations, further validating its efficacy in improving system performance and longevity.

3.7.4 Hardware Overhead and Energy Consumption

Table 3.11 presents the total area overhead of LiveWay for a given buffer size. Overhead comprises of way wise write counters, switch indicator, and LSC apart from the buffer. Since the other hardware components in LiveWay are almost the same as in the baseline, the additional energy consumed by LiveWay (AE_{LW}) can be calculated as $AE_{LW} = E_{buff} - E_{write}$. Where E_{buff} is the energy consumed by the SRAM buffer and E_{write} is

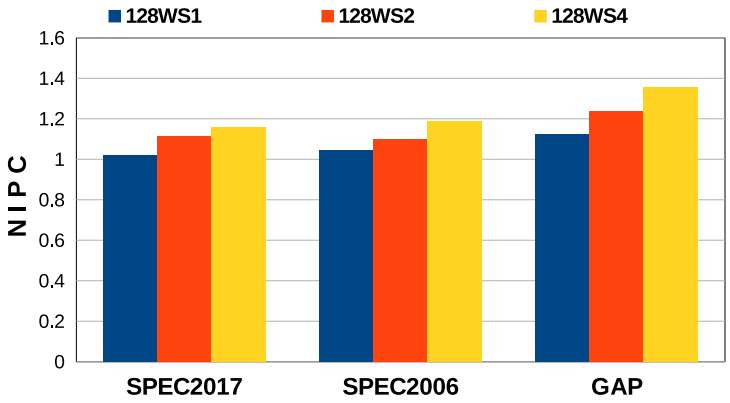

Figure 3.35: Normalized IPC of various different benchmarks run on various configurations.

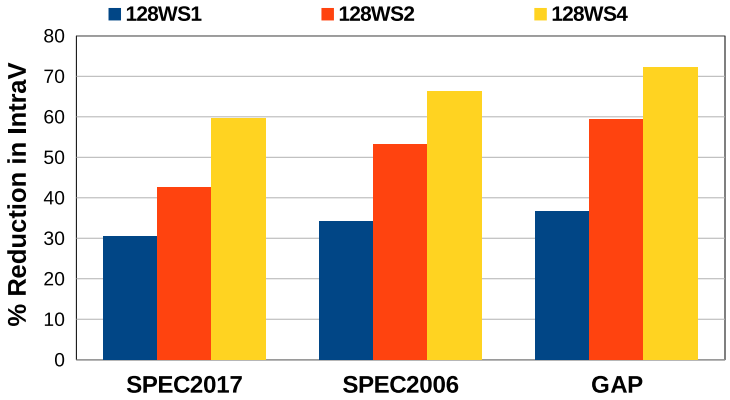

Figure 3.36: Reduction in IntraV of various different benchmarks run on various configurations.

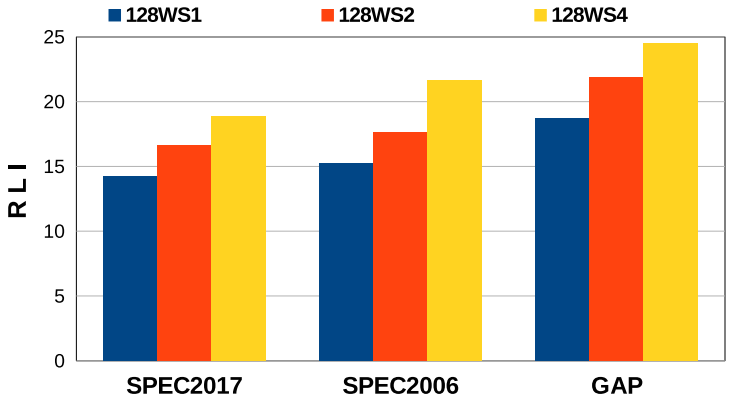

Figure 3.37: RLI of various different benchmarks run on various configurations.

the energy saved by reducing the number of writes. We have used CACTI [1] to model the SRAM buffer. Experimental analysis with all sizes of SRAM buffer found that the value of AE_{LW} is always negative, implying that LiveWay does not have any additional energy overhead with any buffer size, which is enough to get the benefit we are expecting.

Table 3.11 presents the total area overhead associated with LiveWay for a given buffer size. This overhead primarily consists of way-wise write counters, a switch indicator,

and an LSC, in addition to the buffer itself. Since most hardware components in LiveWay remain nearly identical to those in the baseline system, the additional energy consumption introduced by LiveWay (AE_{LW}) can be determined using the following equation:

$$AE_{LW} = E_{buff} - E_{write}$$

where: - E_{buff} represents the energy consumed by the SRAM buffer - E_{write} accounts for the energy saved due to a reduced number of write operations
To model the SRAM buffer, we utilized CACTI [1], a widely used tool for estimating cache and memory energy consumption. Our experimental analysis, conducted across various SRAM buffer sizes, consistently found that the computed value of AE_{LW} was always negative. This result indicates that LiveWay does not introduce any additional energy overhead across different buffer configurations.
These findings are significant, as they confirm that LiveWay achieves the intended benefits without incurring extra energy costs, making it an efficient and practical solution for improving system performance while reducing write-related energy consumption.

Table 3.11: Storage Overhead of LiveWay.

Buffer Size	**Total Overhead**	**Overhead over Baseline**
32KB	48.06KB	1.173%
64KB	80.06KB	1.954%
128KB	144.06KB	3.517%
256KB	272.06KB	6.64%
512KB	528.06KB	12.89%

3.8 Conclusion

Through our work, we have highlighted the need for STT-RAM LLCs over SRAM based LLCs, but implementing purely STT-RAM LLCs comes with its own set of issues like performance degradation and lifetime. Therefore, we propose PROLONG which is an innovative, power-efficient and cost-effective architecture that aims at reducing the InterV and improving the lifetime of the STT-RAM LLC drastically without compromising on the performance of the system. Overall we have achieved a 89% reduction in InterV and 35 times lifetime improvement with a 0-2% improvement in performance for multi-core systems. Therefore, PROLONG becomes a practical alternative to the traditional SRAM based LLC architectures as it can provide both the performance needed and also the required lifetime to implement an STT-RAM based LLC. LiveWay proposed a unique, energy efficient and low cost architecture even when compared to PROLONG. Future research includes proposing wear leveling techniques at block level and bit level granularity.

Chapter 4

Decoupling the tag and data array for Lifetime Improvement

In this chapter, we have proposed a novel wear-leveling technique that distributes the writes across all the blocks in the STT-RAM based LLC. In this technique, a block-wise write counter is implemented in order to count the number of writes per block. Write hot blocks are identified, and then the redirection takes place by exchanging the tag and data array blocks. A hardware efficient approach is also proposed that limits the number of counters by assigning a single counter to a bucket and each bucket consists of a group of blocks.

Publications from this Chapter

- **Prabuddha Sinha**, Krishna Pratik BV, Shirshendu Das and Venkata Kalyan Tavva, "SmartDeCoup: Decoupling the STT-RAM LLC for even Write Distribution and Lifetime Improvement", in Elsevier Journal of System Architecture, February 2025. [DOI: https://doi.org/10.1016/j.sysarc.2025.103367]

4.1 Introduction

Despite the introduction of various techniques and hybrid technologies aimed at improving the lifetime and reducing the write latency of STT-RAM LLCs, as explored in Chapter 2, none have managed to enhance the lifetime of STT-RAM based LLCs very effectively by wear leveling the writes. Majority of the wear leveling techniques were proposed in a set level or way level granularity and none of these techniques aimed at wear leveling in the block level granularity. This prompted us to think of a solution to distribute the writes evenly across all the cache blocks in order to improve the lifetime of the LLC.

4.2 Motivation

In order to analyze the impact of WV, we conduct experiments using a simulator configured for a single-core system with an 8 MB, 16-way set-associative STT-RAM LLC that lacks wear-leveling support and employs a standard LRU replacement policy, for one billion instructions. The experiment utilized *milc* workload, a write-intensive workload. Figure 4.1 clearly depicts the non-uniform write distribution of the write-intensive workload across the whole cache. It shows the write count of each and every cache block present in the cache through the heat map in Figure 4.1. Write patterns are clearly visible for specific sets and blocks. This uneven distribution of writes results in certain regions

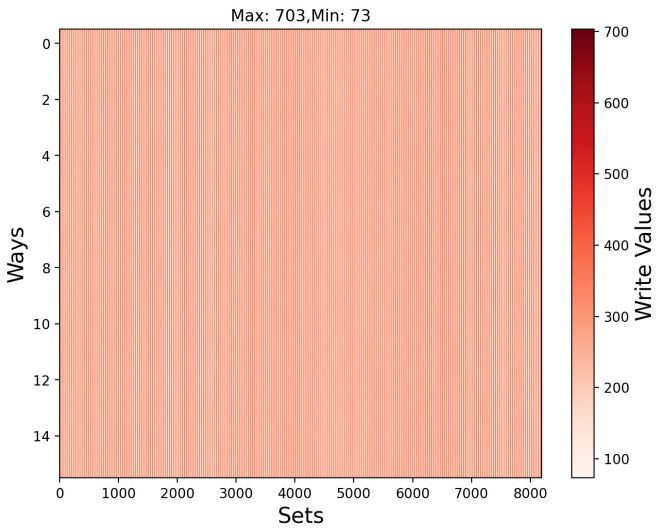

Figure 4.1: Heatmap representing the non-uniform write distribution at LLC for a single-core *milc* run.

of the memory being subjected to a higher frequency of write operations, accelerating the wear and tear in those areas.

As a consequence, the durability of the LLC is compromised, and its effective operational lifespan is shortened. Therefore, write distribution is a critical factor in maintaining the reliability and performance of the memory system. In the case of Figure 4.1 the maximum write count among all blocks in the cache is 703 while the minimum write count is 73 showcasing the huge disparity in the distribution of writes and the urgent need for an efficient block level wear leveling technique, that will help distributes the writes evenly across the blocks. The number of blocks experiencing writes less than 100 is 89234, 30747 blocks experienced writes greater than 500, 10146 blocks experienced writes greater than 600 and only 945 blocks experienced writes greater than 700.

As the core count rises in a cache multiprocessor, so does the number of ways in the cache, consequently increasing its associativity. With increasing associativity in the cache, there's a broader spectrum of options available for implementing block replacement policies. Moreover, heightened associativity results in a more non-uniformly distributed write pattern within the cache.

Furthermore, recent secure cache techniques like Ceaser [70], Ceaser-S [69] and Scatter-Cache [100] proposed redirection techniques in order to improve the LLC security, but it brought along with it extra writes that are non-uniform and will lead to significant reduction in endurance. Ceaser, Ceaser-S and Scatter-Cache bring about a 20%, 5% and 2% increment in remap writes for SPEC CPU 2006 and SPEC CPU 2017 benchmarks. A more detailed discussion about these techniques is given in Section 4.3.5. Although the endurance of an STT-RAM has been enhanced a lot through recent technological advancements [11, 101, 102]. There is still a need for ensuring that the endurance of a STT-RAM based LLC when combined with the above mentioned secure cache techniques. The existing endurance enhancement techniques cannot be directly implemented on top of

these security countermeasures. The existing endurance enhancement techniques cannot be directly implemented on top of these security countermeasures. Recent techniques like POEM [22], ARMOR [103], and logical cache partitioning [39] show that the problem of endurance enhancement in the use of STT-RAM LLCs is still being researched to this day. Hence, we have been motivated to propose SmartDeCoup, which can handle endurance issues of both secure and non-secure STT-RAM LLCs.

4.3 SmartDeCoup

The main goal of this work is to enhance the lifetime of the STT-RAM LLC by reducing WVs. For this, a small trade-off of performance is considered acceptable. To achieve the goal SmartDeCoup introduces a decoupled tag and data array in such a way that each location from the tag array maps to a unique location in the data array, i.e., a 1:1 mapping throughout. The mapping between the tag and data array locations can be interchanged, while consistently maintaining a 1:1 mapping between tag and data array. As mentioned in Section 4.1, the concept of decoupled tag/data array is inspired by V-way [104] and Z-cache [105]. Recently the technique has been reintroduced in Maya cache [106] for enhancing the security of the system. SmartDeCoup has two fundamental differences with these techniques. First, in all the existing decoupled tag/data structures, the number of data array entries is less than the tag array entries. In SmartDeCoup, the number of tag and data entries are equal, and always a 1:1 mapping is maintained. Second, none of the existing techniques use the concept of decoupled tag/data array for enhancing the lifetime of an STT-RAM LLC.

In this chapter we have proposed 2 different techniques that use the concept of decoupling the tag and the data array: *Primal Approach* and *Hardware Efficient Approach*. The major difference between them lies in how writes are redirected from write hot spots to cold spots with the help of swapping or redirection. *Primal Approach* employs the *SWAP* operation from write hot blocks to cold blocks while the *Hardware Efficient Approach* uses redirection from write hot buckets to cold. One more major difference is the number of write counters, while the *Primal Approach* uses one write counter per block the *hardware Efficient Approach* uses one write counter per bucket. The Primal Approach and the Hardware Efficient Approach are both mutually exclusive and cannot work together with each other.

4.3.1 LLC Organization

Figure 4.2 presents the organization of the proposed LLC. It consists of two different structures that have been decoupled: tag array and data array. The tag array is designed with SRAM and the data array is designed with STT-RAM. The tag array has a set-associative structure but the data array can be considered as fully associative because a block can be placed in any where in it. The *tag entry* and *data entry* are the two parts of a block B placed in the tag and data array of the LLC respectively. We use

the term "entry" to represent a cache block and not the location in tag/data array. The location in both tag and data array is represented by (x, y); where x is the set number and y is the way number. A location in tag array always maintains a forward pointer ($FPTR$) that points to a data location in data array. Similarly, a location in data array always maintains a backward pointer ($BPTR$) to point to the corresponding tag array location. Each tag entry contains status bits and tag bits. The status bits consist of a valid bit, a dirty bit, and coherency bits. Each data entry consists of the data and a valid bit. The tag array uses a traditional replacement strategy like LRU. Data array does not require any replacement decision. The reason we consider an SRAM based tag array is because of the faster look up time. Furthermore, tag array generally has more updates because of status bits, replacement policy bits and coherence bits, therefore it is reasonable to implement a volatile memory that does not have lifetime constraints as the tag array [107].

Invalidation: Invalidating a tag entry means the corresponding data entry also needs to be invalidated and vice versa. However, invalidation doesn't destroy the $FPTR$ and $BPTR$ values that are associated with the location and not with the block. Hence, the mapping between the tag and data location remains even after the invalidation. This is a fundamental difference between our decoupled tag/data array structure as compared to the other existing such designs [104] and [105].

Block insertion: To insert a new block (say, B) into the LLC, the tag part of B (tag entry) is placed in the tag array. Since the tag array is set associative, B maps to a particular set (say, S) in tag array. To insert B in S, an existing tag entry may need to be evicted (say, tag entry of block C is the victim) based on the replacement policy. The entry for C is invalidated from both the tag and the data arrays, and the corresponding locations are allocated to B.

Block search: To search a block B in LLC, only the tag array is searched. In case of a tag hit, the corresponding data entry is accessed trough $FPTR$. On a tag miss, the request is forwarded to the main memory and before receiving the block from the main memory an entry for the block is allocated in LLC (both tag and data entry) as discussed above.

4.3.1.1 Maintaining Coherence:

The concept of separate tag and data storage is already used in literature [86, 57]. In the decoupled cache architecture, the tag store serves as the central repository for all metadata associated with cache blocks. This includes critical information such as the replacement policy and the sharers directory, both of which are also managed within the tag storage. The operations of installing new cache blocks and invalidating existing ones within the data storage are guided by the forward pointer maintained at the corresponding tag location. Notably, all key actions—such as invalidations and modifications to the sharer bits—are confined exclusively to the tag storage. If an invalidation is required at the tag level, the associated data entry is invalidated in a seamless manner by following the forward pointer. This strategic separation of tag and data storage ensures that cache coherence is preserved without introducing any inconsistencies, thereby enhancing both efficiency and scalability.

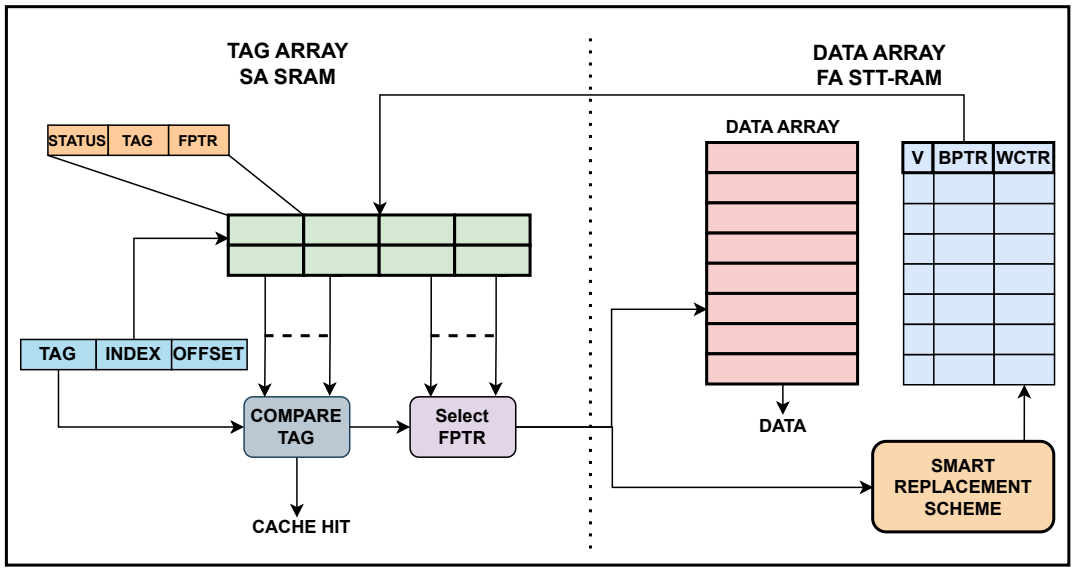

Figure 4.2: The proposed decoupled cache.

4.3.2 Primal Approach

Our proposed *Primal Approach* is based on the underlying decoupled architecture discussed in Section 4.3.1. The main goal of this work is to decouple the tag and data part of the LLC such that efficient wear-leveling is carried out and the writes are evenly distributed among all the locations of the data array. An 8-bit write counter (*WCTR*) is maintained for each data location, which is responsible for keeping track of the write count in each data location. The counters are an extra SRAM array structure that is co-located with the STT-RAM data array. It is incremented on every write to that specific data location. After every epoch, the data entries located in top $\boldsymbol{Q\%}$ heavily written data locations are swapped with the data entries located in bottom $\boldsymbol{Q\%}$ data locations. Sorting such a large number of counters before enabling the `SWAP` brings about a significant computational overhead in the MSHR. The *WCTR* values are not reset after every epoch. If at any time a *WCTR* value saturates then after that epoch the value of that *WCTR* is immediately decreased by half of its maximum value. Swapping a heavily written block (only data entry) with a lightly written block in the data array distributes the writes across the data array. The most heavily written block is swapped with the block that is least written and the second most heavily written block is swapped with the block that is second least written into. This continues on and gradually until all the top $\boldsymbol{Q\%}$ heavily written blocks are swapped with the bottom $\boldsymbol{Q\%}$. The cache pauses during `SWAP` operation and all services are suspended till the operation is completed. The value of $\boldsymbol{Q}$, i.e. the intensity of write hot blocks to be `SWAP` after every epoch has been varied from **5% to 30%** in our experiments.

Swapping operation: When two data entries in the data array swap, the corresponding *BTPR* values are also swapped. In the tag array, only the *FPTR* values are swapped to keep the 1:1 mapping intact. An important point to observe here is that each swapping

needs two writes in the data array and one write in a temporary SRAM buffer. The SRAM buffer is a small buffer that is only used during the `SWAP` operation. Since, all the `SWAP` operations are performed in parallel, the size of the buffer varies with the intensity of the `SWAP` operation, i.e. the value of 'Q'. The periodic swapping may temporarily block the LLC for normal read/write operations. The additional writes caused by `SWAP` and the access latency are considered while implementing SmartDeCoup.

Why the Primal Approach reduces InterV and IntraV? Since the data array logically works as a fully associative cache, any data entry can be swapped with any other data entry. Hence, it can reduce both InterV and IntraV, improving the LLC's lifetime better than the existing techniques. Swapping data entries does not violate the 1:1 mapping between tag and data array as the corresponding *FPTR* and *BPTR* values are also carefully swapped. The main aim of the Primal Approach is to perfectly wear level the cache without worrying about any other overhead.

Why is it called the Primal Approach? This technique is called the Primal Approach because at the specific size of the epoch, this is the best scheme based on which heavily written blocks can be prevented from being written into and writes are evenly distributed to those blocks that are not heavily written. This approach can be considered a platform and be optimized by introducing various other modifications which can be taken up by future research.

4.3.2.1 Working Example

Figure 4.3 highlights the working methodology of the *Primal Approach*. In this study, we consider a compact yet representative Last-Level Cache (LLC) comprising only a limited number of blocks. Initially, there exists a clear and direct 1:1 correspondence between the SRAM-based tag array and the STT-RAM-based data array. This mapping ensures efficient organization and retrieval of cache blocks, establishing a fundamental structure for managing metadata and data storage in tandem. Each tag location has an *FPTR* pointing to a data location and each data location has a *BPTR* pointing to its tag array counterpart. Initially in Step 1, whenever there is an incoming write into a data location, along with the write, the corresponding *WCTR* is incremented. In Step 2, after a predetermined interval has passed the top ***Q%*** and bottom ***Q%*** of the data entries are identified as per their *WCTR* and the blocks that are supposed to be swapped are indicated. In this example, we have considered 2 top and 2 bottom data entries as indicated by red *WCTR* and blue *WCTR* respectively. In Step 3, the `SWAP` operation is initiated and only one `SWAP` operation is shown in Figure 4.3. In Step 3, the `SWAP` operation is completed through a temporary buffer that transfers the *FPTR* and *BPTR* after invalidating the data array location that has *High WCTR*. Step 4 indicates the final location of the *FPTR* and *BPTR* after the whole `SWAP` operation has been performed.

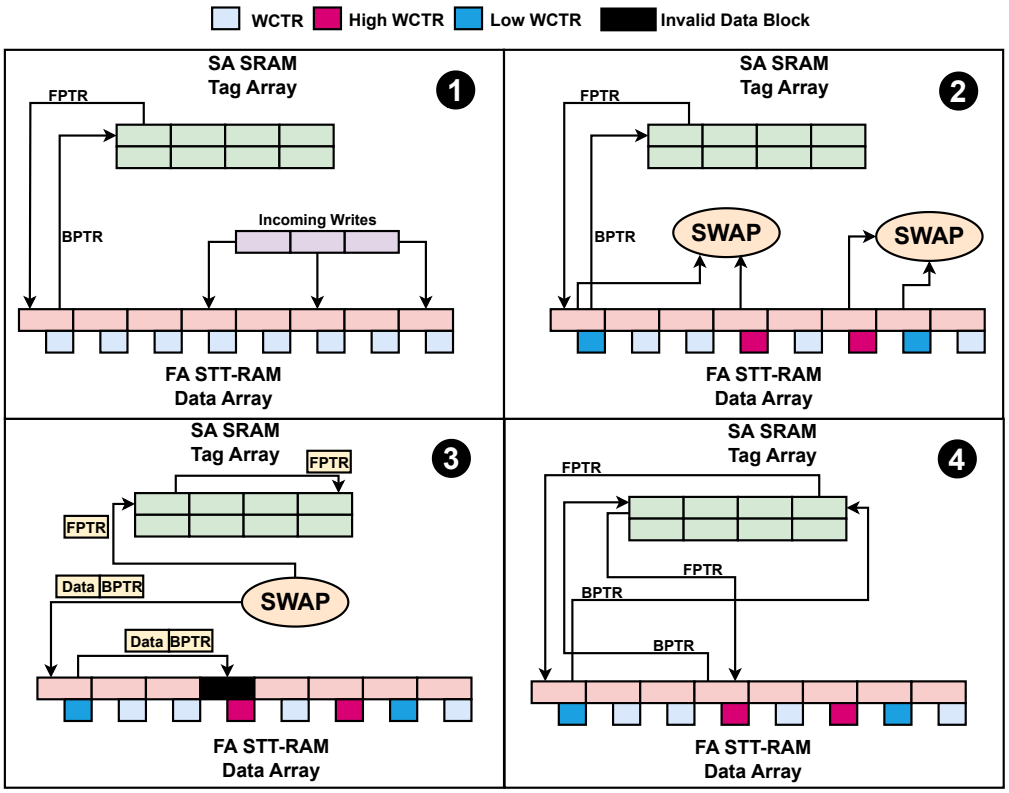

Figure 4.3: Working Example of the Primal Approach.

4.3.2.2 Drawbacks of Primal Approach

The major drawback in the *Primal Approach* is the lack of consideration for the growing hardware and latency overhead with the increase in the intensity of swapping. The *Primal Approach* mainly has three different drawbacks:

- The number of *WCTRs* are a lot, equal to the number of blocks, and therefore the hardware overhead is very high.
- The periodic `SWAP` operations required at the end of every epoch cause the normal operation of the cache to be paused, leading to congestion in the read/write queue of the LLC that may cause significant performance degradation [13].
- The extra energy overhead that is required for the extensive `SWAP` operations. With an increase in the number of `SWAP` operations the energy expenditure goes up.
- The extra area overhead that is incurred due to the SRAM Buffer when the `SWAP` operation is taing place.

4.3.3 Hardware Efficient Approach

In order to counter the drawbacks of the Primal Approach, we propose a *Hardware Efficient Approach* that is aimed at reducing the number of counters by grouping together the data

locations into various buckets. The bucket sizes can vary based on the implementation and there is only a single *WCTR* per bucket. A write in any data block belonging to that bucket causes an increment in its *WCTR*. After every epoch, the bucket with maximum *WCTR* value is made as read-only. The read-only status of a bucket is only for a single epoch. At the end of every epoch, a new read-only bucket is decided and the FPTR and BPTR for all data and tag arrays for the read-only bucket are reset. It is important to note that the buckets are created based on the data locations and not data entries.

How are writes to blocks in read-only buckets handled? If any write request comes for a block B resides in a data location (say L') within the read-only bucket, the write is forwarded to another bucket. A bucket with the least number of writes is selected and from that bucket a location (say L'') is randomly chosen for writing B. If the location L'' currently holds a valid entry then it must be invalidated first. As mentioned previously invalidating a valid data entry needs to invalidate the corresponding tag entry. The *BPTR* values of L' and L'' are swapped and also the corresponding *FPTR* values in the tag array needs to be swapped. Now, the data entry of B is written in the data location L''. The location L' still connects to an invalid tag location. No read-only restrictions are applied to the tag part.

How is a cache miss handled? When there is a miss in LLC and the newly fetched block is about to be written in the read-only bucket, it is allowed to write it in this case. Hence, the read-only restriction is only for write-back from the L2 cache. The reason for this is that if the newly fetched block is read-only then there will be no further write to it. In case, the block is a write-intensive block, then the block will be migrated to another bucket during its next write-back to LLC from upper level of cache.

4.3.3.1 Bucket formation

Each location of the data array must be mapped to a particular bucket. In order to make sure that the mapping is distributed across the whole cache, we apply a hash function which is responsible for assigning the blocks into the buckets. We try to assign the blocks such that buckets are not overly clustered and the mapping to a bucket is relatively distributed across the whole cache. We consider the location of a block in the data array as (x, y), where the set number is x and the column number is y. We prepare a hash function by using these parameters. In our design, we have used the modulus hash function to divide the LLC data blocks into buckets. The main purpose of the hash function is: (i) The data array locations are distributed across the buckets. (ii) Just knowing the location of the data array we can calculate which bucket it belongs to in $\mathcal{O}(1)$ time. Now after the blocks have been distributed into the buckets the system runs as usual with the *WCTR* for every bucket incrementing its value in case of a write to any of its corresponding blocks.

4.3.3.2 Working Example

Figure 4.4 explains the step-by-step process involved in the *Hardware Efficient Approach*. In Step 1, we have specified four buckets B1, B2, B3, and B4. Each bucket has an

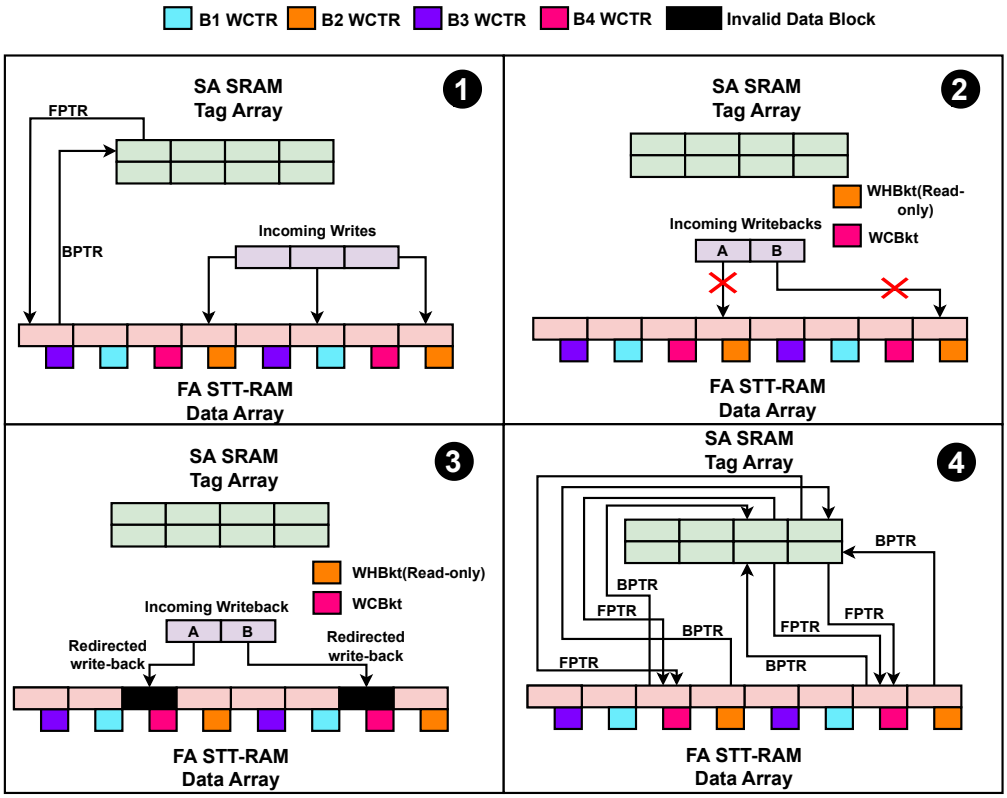

Figure 4.4: Working Example of the Hardware Efficient Approach.

individual *WCTR* as depicted in the figure. The tag and data array entries are connected with a *FPTR* and a *BPTR* respectively. At the start of Step 1 there are no buckets designated as *WHBkt* (Write Hot Bucket) or *WCBkt* (Write Cold Bucket). In Step 2, after the first epoch has passed we identify the bucket having the highest WCTR and assign it as *WHBkt* or read-only bucket. The bucket having the lowest WCTR is assigned as *WCBkt*. Step 2 shows that if we encounter any incoming write-backs A and B in the present epoch to the *WHBkt* then the write-back to that bucket is not possible as it is read-only. Any normal writes that are generated due to read miss is written to the *WHBkt* as usual. Step 3 shows the data array blocks to which the write-backs of the *WHBkt* are redirected-to in the *WCBkt* and the location to where the redirection will happen is selected at random from among the blocks present in *WCBkt*. This randomly selected data block is then invalidated and the redirected write-backs A and B are placed in their respective random locations. The write-back replaces the data block in a random location belonging to *WCBkt*, especially if the data stored is invalidated. Step 4 shows how the different FPTRs and BPTRs are migrated, pointing to different locations in the tag array in the affected locations of the *WCBkt* and the effect of write redirection on the data value and the BPTR. It highlights how for that epoch for any incoming write-back to the *WHBkt* that location in the *WHBkt* loses its FPTR and any new incoming writes-backs are automatically redirected to the corresponding *WCBkt* location that is directed by the FPTR. We have introduced a threshold where the difference between the highest and

lowest WCTR will have to be a minimum 100 writes in the case of the Hardware Efficient Approach, the threshold is dynamically controlled, i.e. if the bucket size is larger the threshold increases with increasing bucket size and decreasing number of buckets. Only if the threshold is met, then there is redirection from the write hot bucket to the write cold bucket. The swapping threshold is clearly indicated in Table 4.3.

4.3.4 Primal Approach vs Hardware Efficient Approach

The *Primal Approach* reduces the IntraV and InterV better than the *Hardware Efficient Approach* as it can uniformly distribute writes across the data array. The major difference between Primal and *Hardware Efficient Approach* is that the former needs periodic `SWAP` operations which incurs extra writes while the latter has no additional writes incurred due to the `SWAP` operation. Each `SWAP` operation needs two additional writes in the data array. In *Hardware Efficient Approach* forwarding a write request from a read-only bucket to another bucket doesn't require any additional write except the compulsory write required for the write-back. However, the *Hardware Efficient Approach* requires invalidation of a block from the LLC to handle a write-back to a read-only bucket. Such invalidation slightly reduces the hit rate of LLC but this reduction is compensated by the zero additional writes (due to swap), potentially eliminating the congestion on the read/write queue of the STT-RAM LLC. The Primal Approach outperforms the Hardware Efficient Approach to some extent and the difference is not so drastic. Section 4.5.3 presents a detailed sensitivity analysis and a thorough comparison between Primal and Hardware Efficient Approach. Furthermore, it is evident that in Primal Approach we employ the use of upto 8K counters that is responsible for keeping track of the write counts per block. Sorting these counters will obviously bring in an overhead, we consider a scenario where the sorting is done in parallel in the MHSR, but this will incur a lot of extra hardware. Therefore, there is a need for a more polished solution like the Hardware Efficient Approach that is feasible.

4.3.5 Importance of SmartDeCoup in Modern LLC

SmartDeCoup has three important benefits over modern LLCs:

- It makes STT-RAM more realistic to use as an LLC by enhancing its lifetime.
- By making the data array as fully associative, it gives a platform to do further optimizations in future.
- It can be used with secure LLCs without any major modifications.

The importance mentioned in the first two points have already been discussed earlier in this chapter. This section focuses on the significance of SmartDeCoup in secure LLC design. LLCs are vulnerable to various timing channel attacks [42], such as side-channel [108] and covert-channel attacks [109, 82]. To prevent these attacks, multiple strategies have been proposed, including cache partitioning [110], randomization [69], and attack detection

[111], etc. Among these countermeasures, the two most recent and secure techniques are Mirage [63] and Maya-Cache [106]. Both techniques are based on a decoupled tag and data array structure, similar to SmartDeCoup. However, the tag arrays in Mirage and Maya-Cache contain more tag locations than data locations. This increased number of tag locations helps to eliminate conflict misses in the cache, which is a key factor in timing channel attacks.

In addition to improving the lifetime, a future STT-RAM LLC must also be secure. So far, little attention has been given to how recent security countermeasures can be integrated with STT-RAM LLCs. The decoupled tag/data array-based countermeasures mentioned above were mainly designed for SRAM-based LLCs. Existing lifetime enhancement techniques like PoLF, WAD, DWAWR, and i^{2}WAP cannot be implemented on top of such designs due to structural mismatches and security concerns. However, SmartDeCoup is built on the same decoupled tag/data array structure used by Mirage and Maya-Cache for securing the cache. Therefore, it can be implemented on top of these secure LLC designs, making SmartDeCoup a better option for LLC design.

4.3.5.1 Challenges in Implementing SmartDeCoup on Mirage and Maya-Cache:

A detailed discussion of Mirage [63] and Maya-Cache [106] is beyond the scope of this paper. The core concept of these techniques is to avoid any evictions caused by conflict misses in the tag array. The *Hardware Efficient Approach* of SmartDeCoup cannot be directly applied to these techniques, as it may cause some invalidation. However, a modified version of this approach that only involves swapping, or the *Primal Approach*, can be used without compromising security. Other wear leveling techniques still consider the cache as a whole set associative cache, while to the best of our knowledge, SmartDeCoup is the first architecture that proposed wear leveling in a decoupled setting with a 1:1 mapping. The *Primal Approach* is compatible with Mirage and Maya-cache as they also implement a decoupled fully associative data array with a set associative tag array. Mirage and Maya-cache provide security but do not keep in mind the endurance, as in the data array they follow a global random replacement policy with some slight modifications. The randomization of block replacement can create unwanted data patterns detrimental to endurance. Hence, SmartDeCoup can enhance the endurance of secure STT-RAM LLCs, a capability that existing endurance techniques cannot achieve.

4.3.5.2 SmartDeCoup with non-decoupled secured LLC designs

The secure LLC design with a decoupled tag and data array is one of the most efficient secure designs. The proposed SmartDeCoup can be integrated with these designs. Non-decoupled secure NVM LLCs can be categorized into two types:

- **Randomized NVM LLC** – This design faces issues due to additional remapping writes. Using a decoupled tag and data array can significantly reduce these costs by

performing the remapping only at the tag array. However, this concept was already introduced by another researcher of our lab.

- **Partitioned NVM LLC** – This design struggles with endurance issues because existing endurance techniques (e.g., SWWR, DWAWR) cannot be directly applied. To address this, SmartDeCoup's decoupled tag and data array design can be used to maintain partitions only in the tag array, while keeping the data array free for the endurance techniques introduced in SmartDeCoup.

4.4 Experiments

4.4.1 Simulator Setup

For our evaluation of SmartDeCoup we use a custom-modified version of the ChampSim simulator [97]. Our experiments are done on both single-core and multi-core systems (primarily quad-core), featuring a three-level cache hierarchy. The L1 and L2 are SRAM-based private caches. In the L3 (LLC) cache, the data array is designed using STT-RAM while the tag array is designed using SRAM. The baseline ChampSim simulator is extensively modified to support the decoupling of tag and data array as depicted in Figure 4.2. Table 4.1 provides the detailed parameters of the system on based on which the simulator is configured. We conduct experiments with three different sizes of LLC: 4MB, 8MB and 16MB. There has been no influence on the L2 cache because of the LLC modifications. The L2 works as it is, all the changes are at the LLC level from the cache decoupling to the redirection and swaps. In our work we have considered only the LLC as an STT-RAM all other memory structures are unchanged. We have used the read and write latencies for implementing STT-RAM based LLC as the same as proposed by Korgaonkar et al. [13], this is because using a hybrid SRAM and STT-RAM will have the same latency as a normal STT-RAM based cache as shown by Coi et al. [107].

Table 4.1: Simulation parameters of the baseline single-core system.

System Components	Parameters
Core	Out-of-order, bimodal branch predictor, 4 GHz with 6-issue width, 4-retire width, 352-entry ROB
L1I	32 KB, 8-way, 4 cycles
L1D	48 KB, 12-way, 5 cycles
L2	1MB 8-way associative, 10 cycles, LRU
LLC	4MB STT-RAM, 16-way, RL: 20 cycles, WL: 100 cycles, LRU
MSHRs	8/16/32 at L1I/L1D/L2, 64/core at the LLC
DRAM controller	64-entry RQ and WQ, reads prioritized over writes, Burst write: 6/8th of queue size
DRAM chip	4KB row-buffer per bank, open page, burst length 16, t_{RP}: 12.5ns, t_{RCD}: 12.5ns, t_{CAS}: 12.5ns, t_{RAS}:12.5ns, Scheduling Policy:FRFCS

We conduct a comprehensive comparison of our approach with several state of the art techniques including PoLF [12], WAD [15], SwS [12], G-OAS [15], i^{2}WAP [12], Sequoia [15], SWWR [14], DWAWR [14], along with the baseline STT-RAM LLC. The baseline employs an LRU replacement policy and does not incorporate any wear-leveling mechanism. In PoLF, the flush threshold (FT) is configured to 16 for our simulations. WAD employs the clean-LRU block replacement algorithm for cache management. It utilizes a 3-bit saturating counter (SC) to track writes. SwS focuses on swapping the mappings of two sets at a time and shifting the remaining sets based on the swap threshold (ST). G-OAS organizes the cache sets into small groups (such as 4, 8, or 16) and facilitates the swapping of write-hot sets with cold sets within these groups. SWWR employs a window size of four ways and utilizes a 12-bit write counter for each window to track write operations. In contrast, DWAWR uses a 11-bit write counter per way to count the writes.

We experiment with LLC size: 4MB, 8MB, and 16MB with four different configurations of *Primal Approach* and *Hardware Efficient Approach* respectively as depicted in Table 4.2. For our experiments, we define the epoch size as 10^5 instructions. In a single-core setup for each workload, the warm-up phase consists of 100 million instructions followed by 1 billion instructions for execution. In the case of a multi-core setup, specifically for our quad-core systems, the simulation initiates with a warm-up phase of 50 million instructions followed by a run of 250 million instructions per core to complete the workload.

We have implemented a threshold where the difference between the highest and lowest WCTR will have to be a minimum of 20 writes in the case of the Primal Approach, while in the case of the Hardware Efficient Approach the threshold is dynamically controlled, i.e. if the bucket size is larger the threshold increases with increasing bucket size and decreasing number of buckets. Therefore, the lower the number of buckets, the higher the switching threshold. Table 4.3 shows us the different switching thresholds for the different bucket count.

Table 4.2: Configurations of *Primal* and *Hardware Efficient Approaches*.

No.	**Primal Approach**	**Hardware Efficient Approach**
1	Top 5%SWAP	2 buckets
2	Top 10%SWAP	10 buckets
3	Top 20%SWAP	26 buckets
4	Top 30%SWAP	100 buckets

Table 4.3: Thresholds for Switching in different bucket count for *Hardware Efficient Approaches*.

Hardware Efficient Approach	**Redirection Threshold**
2 buckets	2500
10 buckets	1000
26 buckets	500
100 buckets	100

4.4.2 Workloads

We use write-intensive benchmarks from the SPEC2006 and SPEC2017 benchmark suite [98] for our experiments. To identify these benchmarks, we simulated the full range of SPEC2006 workloads to pinpoint those that demonstrated write-intensive characteristics within the LLC. We also incorporate several write-intensive graph applications from the GAP benchmark suite [99] into our simulations. Table 4.4 presents a comprehensive list of all the write-intensive workloads utilized in our single-core simulations. We have also employed the use of AI workloads and analyzed them in Section 4.5.3.3. For the quad-core setup, we employ three distinct workload mixes, each chosen with varying characteristics to capture a broad spectrum of write patterns. The specific composition and attributes of these workload mixes are detailed in Table 4.5. Here, L, H and G denotes the characteristics of those workloads, denoting low write intensive, high write intensive and graph workloads respectively. We have also conducted a detailed comparison between the SPEC2017 [112], SPEC2006 [98] and the GAP benchmark suite [99] in Section 4.5.3.2. We have also run our simulations on AI workloads like deepsjeng, leela, and exchange2 in order to capture how they behave with our proposed techniques.

Table 4.4: Write Intensive Workloads for single-core system.

Benchmark	**Write Intensive Workloads**
SPEC2006	cactusADM_734B, gcc_13B, GemsFDTD_109B, lbm_94B, leslie3d_94B, libquantum_964B, mcf_46B, milc_360B, soplex_66B, sphinx3_883B, wrf_1212B, xalancbmk_99B, zeusmp_100B
SPEC2017	cactusADM_1804B, gcc_16B, GemsFDTD_765B, lbm_1274B, leslie3d_134B, libquantum_714B, mcf_22B, milc_127B, soplex_92B, sphinx3_234B, wrf_575B, xalancbmk_10B, zeusmp_10B
GAP	bc-12, bfs-10, cc-13, pr-5, sssp-5
AI Workloads	deepsjeng, leela, and exchange2

Table 4.5: Workloads for Quad-Core System.

Mix Type	**Workloads**
Mix1-LLHH	gobmk, gromacs, mcf, libquantum
Mix2-HHHH	mcf, libquantum, lbm, xalancbmk
Mix3-GGGG	pr (page rank), bc (betweenness centrality), bfs (breadth first search), sssp (single source shortest path)

4.5 Results and Analysis

In this section, we present a comparative analysis of various wear-leveling techniques against the baseline STT-RAM LLC, evaluating them across a range of metrics. We also present a detailed comparison of our proposed techniques against these state-of-the-art techniques. All the results here are shown for a 4MB LLC, the 8MB and 16MB LLC follow similar trends.

4.5.1 Primal Approach

In order to show the effectiveness of our Primal Approach we have shown how the write distribution is spread across a 4MB, 16-way set associative LLC. Here, the number of writes per epoch of 100,000 instructions is depicted for the SWAP10% configuration. Figure 4.5 for Epoch 1 shows the write concentration on one side of the cache which is then remapped to another side where Epoch 2 in Figure 4.6 shows concentration, Epoch 3 in Figure 4.7 shows the concentration in the central areas. Therefore, from these heat-maps it is clearly evident how writes are distributed across the LLC epoch wise and the wear leveling is done effectively.

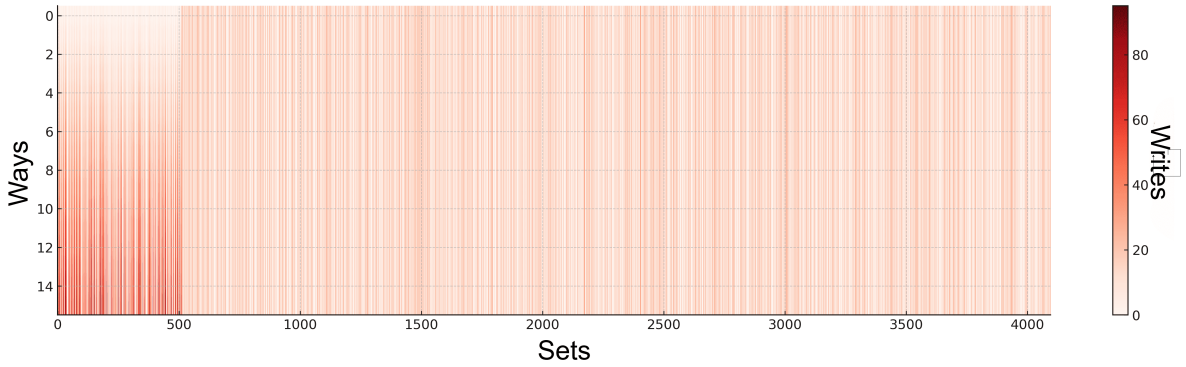

Figure 4.5: Write Distribution for Epoch 1 for *Primal Approach on SWAP 10%*.

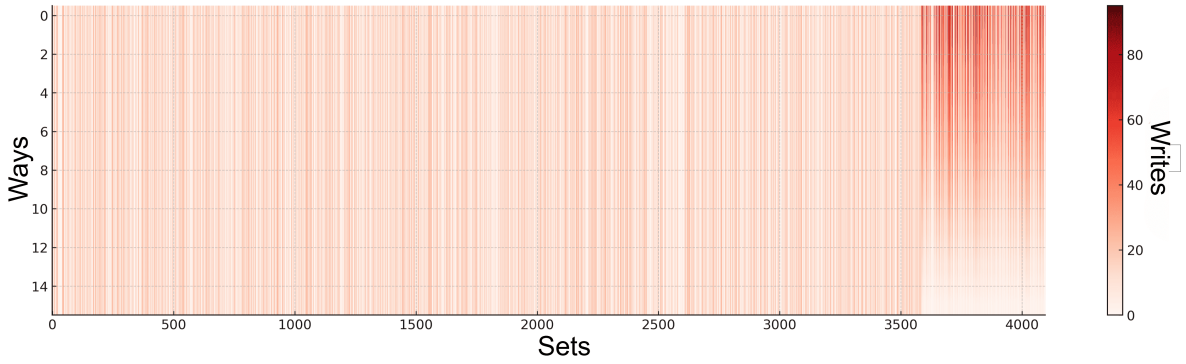

Figure 4.6: Write Distribution for Epoch 2 for *Primal Approach on SWAP 10%*.

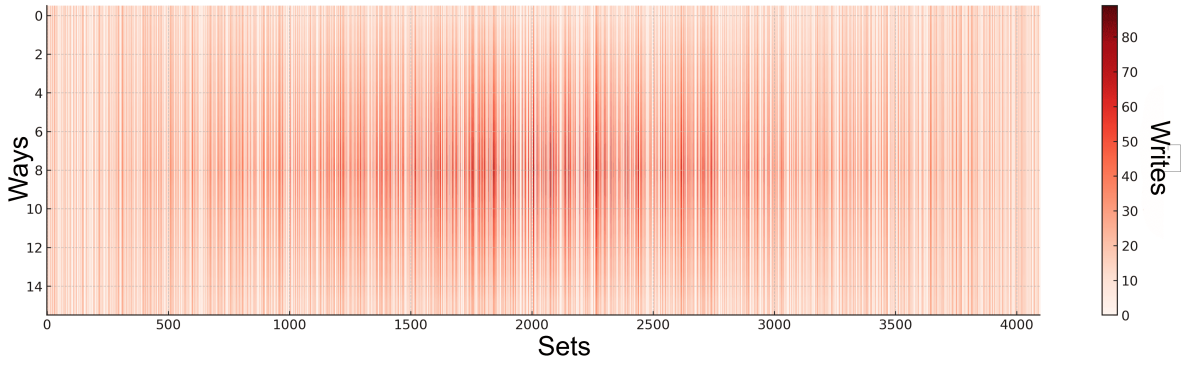

Figure 4.7: Write Distribution for Epoch 3 for *Primal Approach on SWAP 10%*.

4.5.1.1 Single-core Analysis

All the configurations of the *Primal Approach* are compared with the state-of-the-art wear leveling techniques. Figure 4.8 shows us the RLI in times as compared to the other state-of-art in a single-core system with write-intensive benchmarks. It can be observed that the 30% SWAP brings about a 14.45 times improvement in lifetime, the highest relative lifetime improvement as compared to the other techniques for STT-RAM LLC. The trend of reduction in RLI with decreasing %SWAP is also clearly evident. 14.45×, 13.68×, 10.80×, and 6.42× depicts the geomean values for RLI of 30%SWAP, 20%SWAP, 10%SWAP and 5%SWAP respectively.

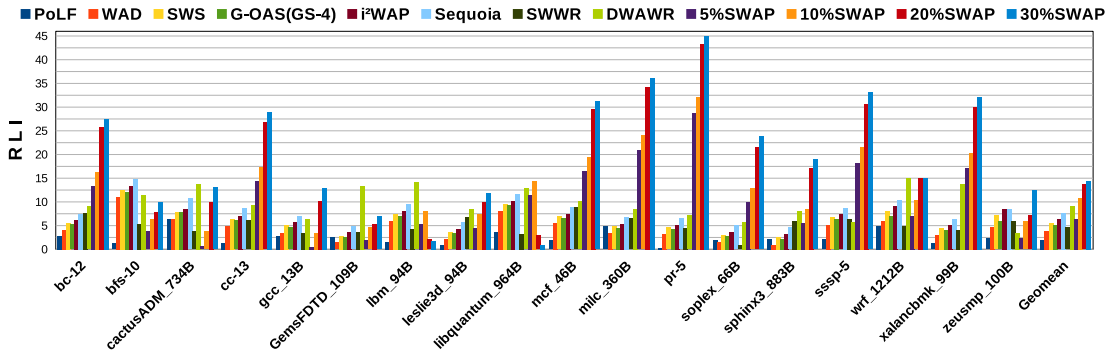

Figure 4.8: Relative Lifetime Improvement in single-core systems for *Primal Approach*.

Figure 4.9 illustrates the reduction in IntraV achieved by various wear-leveling techniques, normalized against the baseline. It can be observed that the 30% SWAP with *Primal Approach* provides the highest reduction in InterV of 80%. This represents a significant advancement over the previous state-of-the-art methods, specifically, a 15% reduction with PoLF and a 29% reduction with WAD, both of which are specialized techniques focused on reducing IntraV. Our *Primal Approach* demonstrates superior effectiveness in mitigating IntraV, further enhancing the reliability and longevity of the system. Figure 4.10 brings to the limelight that the specific techniques like SWS and G-OAS(4) aimed at reducing only InterV achieve a reduction of 89% and 78% respectively. Meanwhile the 30% SWAP of the *Primal Approach* could also garner a respectable 83% reduction in InterV. i^{2}WAP(PoLF+SWS) and Sequoia(WAD+G-OAS) are a combination of 2 different techniques aimed at reducing both the IntraV and InterV simultaneously. Figures 4.9 and 4.10 depict a significant reduction in InterV and IntraV for these techniques. However, our proposed *Primal Approach* provides a block level wear leveling technique that is the best of both worlds and reduces both the InterV and the IntraV quite significantly. It is also clearly evident that the reduction in WVs increases with an increase in the SWAP percentage, further analysis and details of this trend are done in Section 4.5.3.

Figure 4.11 demonstrates that despite the substantial overhead associated with wear-leveling at the block level, the impact on system performance remains minimal. This is reflected in the normalized IPC values for the *Primal Approach* that show only a slight performance degradation. The normalized IPC of 5%SWAP, 10%SWAP, 20%SWAP and

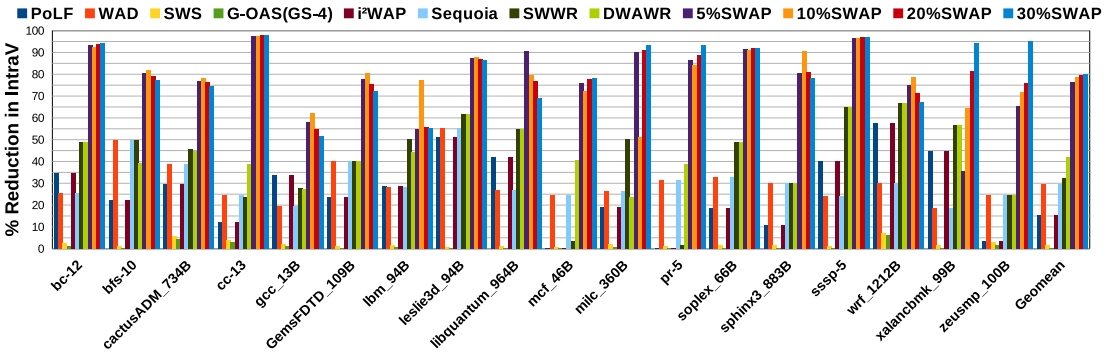

Figure 4.9: Percentage reduction of IntraV in single-core systems for *Primal Approach*.

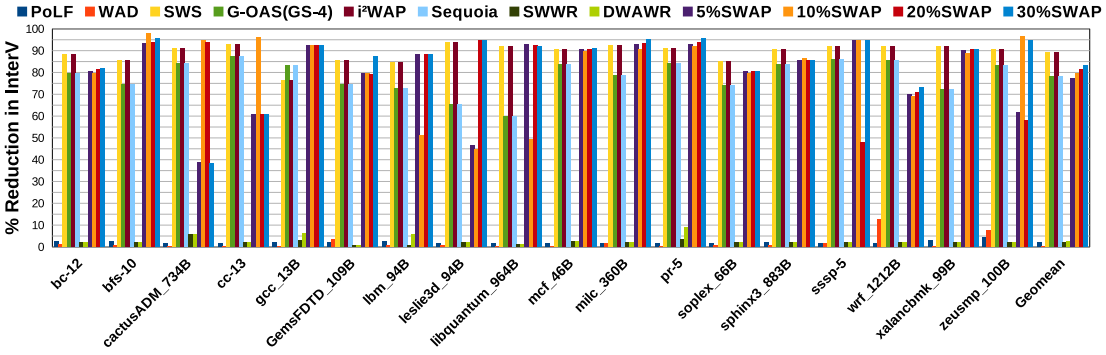

Figure 4.10: Percentage reduction of InterV in single-core systems for *Primal Approach*.

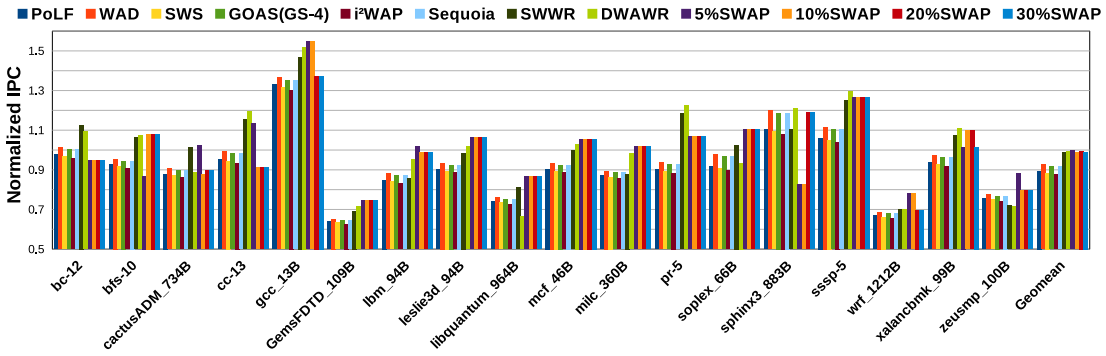

Figure 4.11: Normalized IPC in single-core systems for *Primal Approach*.

30%SWAP are 0.997, 0.986, 0.994, and 0.990 respectively. This indicates that the *Primal Approach* effectively balances wear leveling with performance, maintaining efficiency even under high-overhead conditions that occur due to extensive SWAP operations in larger configurations like 30% SWAP.

Table 4.6 summarizes all the values that have been obtained for different configurations of the Primal Approach compared to the state-of-the-art techniques. It shows that how much more effective our Primal Approach is in improving the Lifetime of the STT-RAM based LLC with minimal impact on the performance.

Table 4.6: Geomean Comparisons for Single-core Systems with *Primal Approach*.

Technique	RLI	% red IntraV	% red InterV	NIPC
PoLF	11.93	14.98	2.04	0.89
WAD	11.23	29.34	0.42	0.92
SWS	12.18	1.82	86.57	0.88
G-OAS(GS-4)	11.94	0.39	78.07	0.91
i²WAP	17.14	14.98	86.57	0.87
Sequoia	17.76	29.34	78.07	0.91
SWWR	5.67	33.22	2.01	0.98
DWAWR	13.55	40.85	2.49	0.99
5%SWAP	32.96	76.54	77.39	0.99
10%SWAP	34.98	78.87	79.76	0.98
20%SWAP	40.97	79.71	81.26	0.99
30%SWAP	44.55	80.13	83.44	0.99

4.5.1.2 Multi-core Analysis

The multi-core experiments exhibit behavior similar to that observed in the single-core experiments, but with effects that are amplified in scale. This amplification highlights the increased complexity and challenges of managing wear leveling and performance in a multi-core environment, while still reflecting the underlying trends identified in the single-core analysis. Figure 4.12 shows that RLI for 5%SWAP, 10%SWAP, 20%SWAP and 30%SWAP is 5.445×, 7.657×, 9.103×, and 12.347× respectively. This is significantly higher than the state-of-the-art, a trend similar to single-core runs.

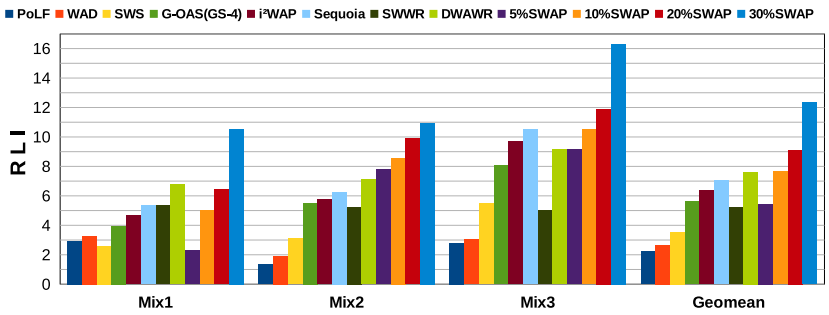

Figure 4.12: Relative Lifetime Improvement in multi-core systems for *Primal Approach*.

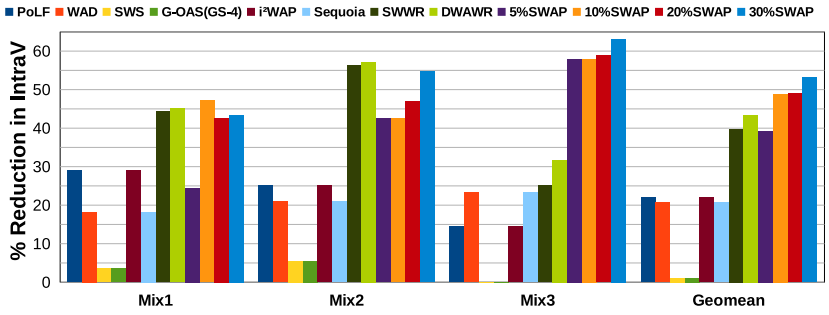

Figure 4.13: Percentage reduction of IntraV in multi-core systems for *Primal Approach*.

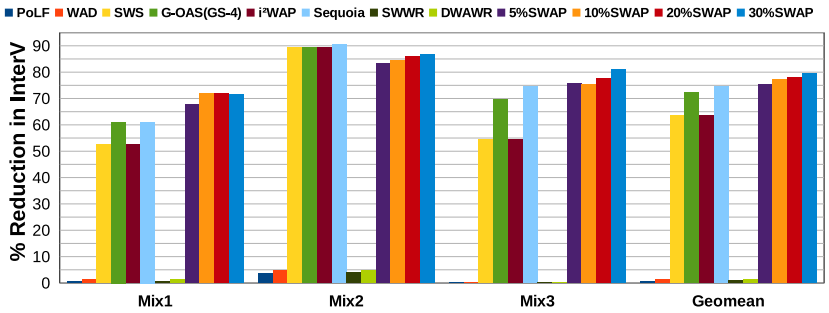

Figure 4.14: Percentage reduction of InterV in multi-core systems for *Primal Approach*.

Figures 4.13 and 4.14 present a comparative analysis of various wear leveling techniques, focusing on their effectiveness in reducing IntraV and InterV, respectively. The reduction in IntraV for 30% SWAP of *Primal Approach* in the multi-core system is 53% which is much lower than the reduction in the single-core system of 83% in 30% SWAP. This is because of the frequently changing diverse data write patterns in multi-core systems. In the case of InterV the reduction values are similar for single-core (83%) and multi-core (79%) for 30% SWAP in the *Primal Approach*.

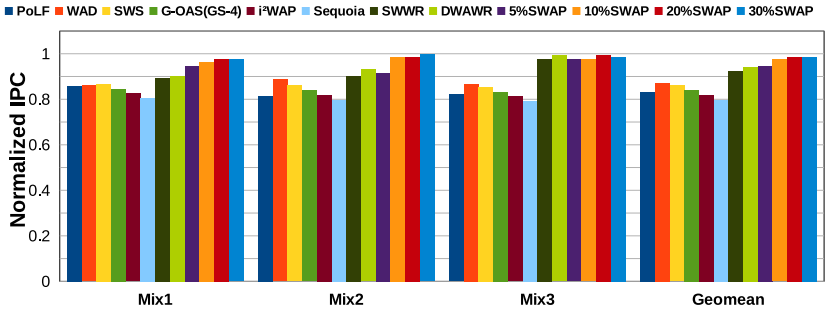

Figure 4.15: Normalized IPC in multi-core systems for *Primal Approach*.

The performance of multi-core systems follows similar trends as single-core systems as depicted through Figure 4.15, but the different Mixes show varying degrees of responses to different wear leveling techniques. For example, Mix1 is a mixture of both read-intensive and write-intensive workloads and they show a reduced IPC as compared to other Mixes due to wear leveled writes taking up read locations. The maximum normalized IPC is observed in the case of 30% SWAP, which is an improvement over the state-of-the-art techniques. A single trend runs through all these figures, showing that the higher % SWAP configurations achieve better improvements as compared to others. This is because with increasing levels of wear-leveling in write-intensive workloads the significant blocks stay in the LLC longer and they can be serviced when any requests come for those blocks and it need not incur the miss penalty.

Table 4.7 summarizes all the values that have been obtained for different configurations of the Primal Approach compared to the state-of-the-art techniques. It shows that how much more effective our Primal Approach is in improving the Lifetime of the STT-RAM based LLC with minimal impact on the performance.

Table 4.7: Geomean Comparisons for Multi-core Systems with *Primal Approach*.

Technique	RLI	% red IntraV	% red InterV	NIPC
PoLF	10.47	21.96	0.7	0.83
WAD	10.31	20.7	1.17	0.87
SWS	15.93	0.93	63.56	0.86
G-OAS(GS-4)	14.79	1.08	72.39	0.83
i²WAP	21.37	21.96	63.5	0.81
Sequoia	22.49	20.7	74.42	0.79
SWWR	8.09	39.71	0.83	0.92
DWAWR	19.49	43.41	1.27	0.94
5%SWAP	25.83	39.21	75.20	0.94
10%SWAP	28.74	48.83	77.04	0.97
20%SWAP	30.14	48.9	78.12	0.98
30%SWAP	33.41	53.13	79.42	0.98

4.5.2 Hardware Efficient Approach

4.5.2.1 Single-core Analysis

Figure 4.16 shows us the increment in RLI as compared to the state-of-the-art techniques. It can be seen that the configuration with 2BKT shows the highest RLI of 13.07× as compared to a maximum of 9.14 times achieved by DWAWR. For 10BKT, 26BKT and 100BKT we observe RLI of 9.65×, 8.80×, and 6.008× respectively. Reduction in IntraV and InterV is highlighted in Figures 4.17 and 4.18 respectively. It is evident that the proposed technique achieves a very high reduction in IntraV of nearly 82% in the case of 2BKT, while the other best technique DWAWR provides a reduction of 42%. For InterV, there is a reduction of 88% in the case of 2BKT which is comparable to the best state-of-the-art technique SWS that provides a reduction of 89%.

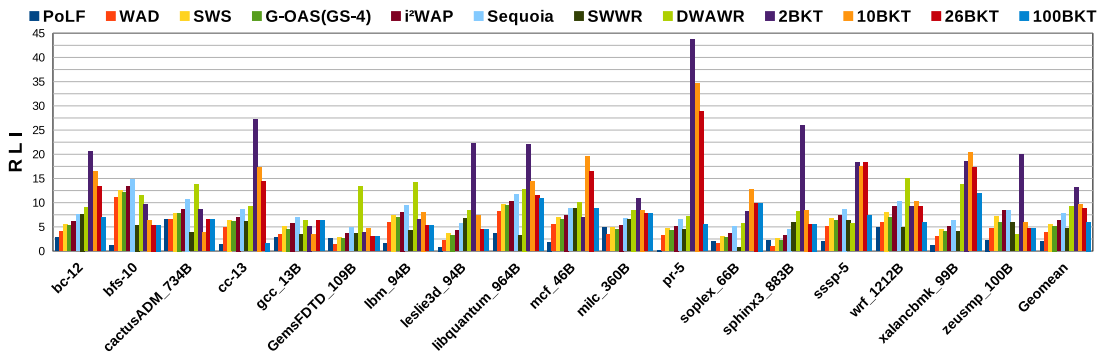

Figure 4.16: Relative Lifetime Improvement in single-core systems for *Hardware Efficient Approach*.

Furthermore, the trend of increasing RLI and increased reduction in IntraV and InterV with a decrease in the number of buckets is evident. This is because the lower the number of buckets, the greater the size of the hot bucket and the more the number of redirections. For example, if we have two buckets then half the LLC is considered as Write Hot Bucket, which will lead to all writes to that section of the LLC to be redirected to the Write Cold

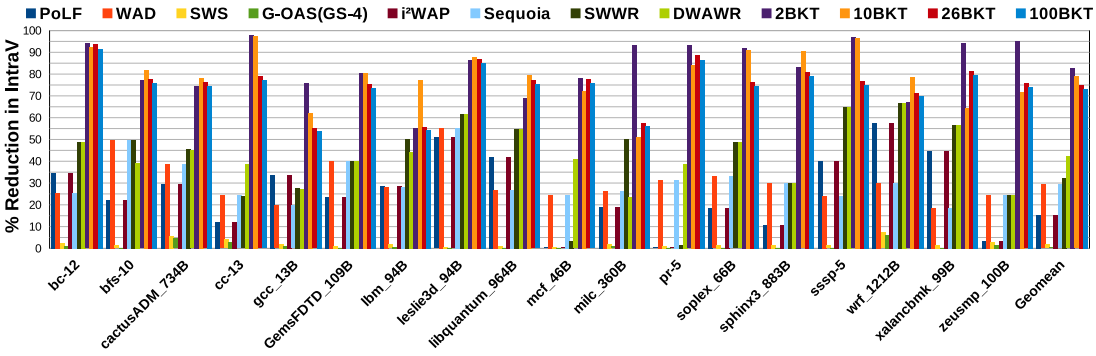

Figure 4.17: Percentage Reduction of IntraV in single-core systems for *Hardware Efficient Approach.*

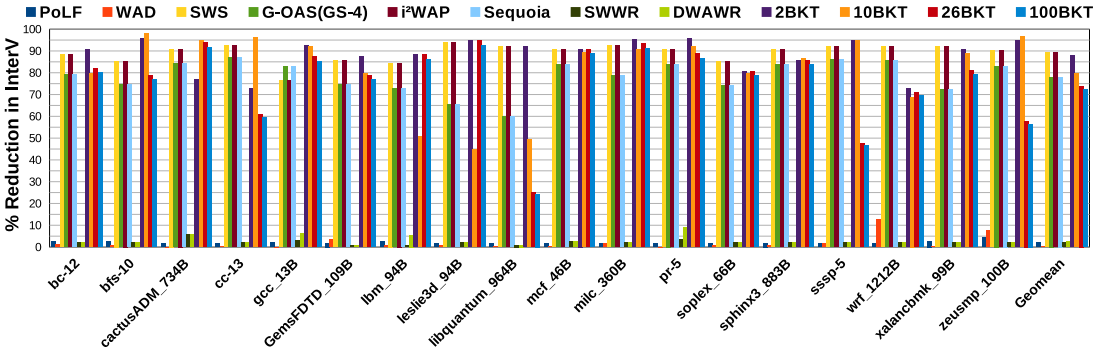

Figure 4.18: Percentage Reduction of InterV in single-core systems for *Hardware Efficient Approach.*

Bucket, i.e. the other half of the LLC. Now increasing the number of buckets will lead to a smaller Write Hot Bucket and therefore lower the number of redirections. This is mainly because with the number of buckets decreasing, there is an increase in the number of redirections, which causes empty locations to appear in the read-only write hot bucket that are now labeled with dirty bits. Therefore, the lower the bucket count, the more the number of redirections, and the lower the hit rate of the cache because of invalidations as the amount of data stored in the LLC is lower, which in-turn is responsible for the performance degradation.

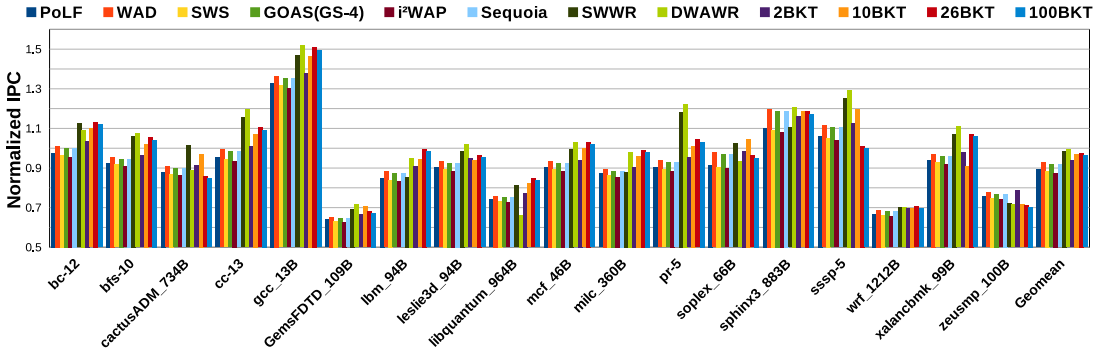

Figure 4.19: Normalized IPC in single-core systems for *Hardware Efficient Approach.*

In Figure 4.19 we observe a small degradation in performance with the 2BKT configuration

having the lowest normalized IPC of 0.93 which is still better than all the proposed state-of-the-art techniques. These results suggest that the *Hardware Efficient Approach* successfully balances wear leveling with system performance, maintaining efficiency even in the presence of significant redirection overhead that occurs mainly in 2BKT configuration. Table 4.8 summarizes all the values that have been obtained for different configurations of the Primal Approach compared to the state-of-the-art techniques. It shows that how much more effective our Hardware Efficient Approach is in improving the Lifetime of the STT-RAM based LLC with minimal impact on the performance.

Table 4.8: Geomean Comparisons for Single-core Systems with *Hardware Efficient Approach*.

Technique	RLI	% red IntraV	% red InterV	NIPC
PoLF	11.93	14.98	2.04	0.89
WAD	11.23	29.34	0.42	0.92
SWS	12.18	1.82	86.57	0.88
G-OAS(GS-4)	11.94	0.39	78.07	0.91
i^{2}WAP	17.14	14.98	86.57	0.87
Sequoia	17.76	29.34	78.07	0.91
SWWR	5.67	33.22	2.01	0.98
DWAWR	13.55	40.85	2.49	0.99
2BKT	42.36	82.63	88.09	0.93
10BKT	34.96	78.57	79.76	0.96
26BKT	31.57	74.95	73.93	0.97
100BKT	28.97	73.38	72.13	0.96

4.5.2.2 Multi-core Analysis

The multi-core setup reveals behavior similar to single-core experiments but with amplified effects, highlighting increased complexity, challenges in wear leveling, and performance management, while retaining core insights on scaling impacts. Figure 4.20 illustrates the impact of wear leveling techniques on extending the multi-core system's lifetime, comparing state-of-the-art methods with the proposed *Hardware Efficient Approach*. The 2BKT configuration achieved the highest relative lifetime improvement of 12.34× , while the next best technique, DWAWR achieved only 7.61×. Figures 4.21 and 4.22 provide a comparative analysis of wear-leveling techniques, focusing on reducing IntraV and InterV respectively. The 2BKT configuration of the proposed *Hardware Efficient Approach* achieves a maximum IntraV reduction of 51%, outperforming DWAWR's 43%, though significantly lower than the 82% reduction seen in single-core systems. For InterV reduction, the 2BKT configuration achieves a 75% reduction, slightly better than Sequoia's 74% but lower than 88% obtained by 2BKT for single-core systems. The normalized IPC values for the 2BKT is relatively lower than its counterparts because in case of 2BKT we see a large number of redirections that will in turn effect the search time and the performance of the system.

Table 4.9 summarizes all the values that have been obtained for different configurations

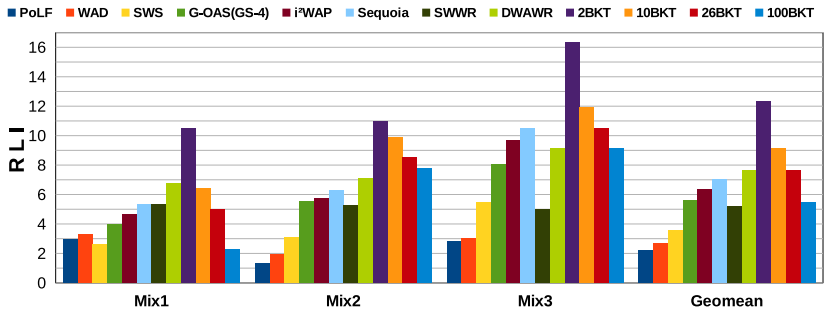

Figure 4.20: Relative Lifetime Improvement in multi-core systems for *Hardware Efficient Approach*.

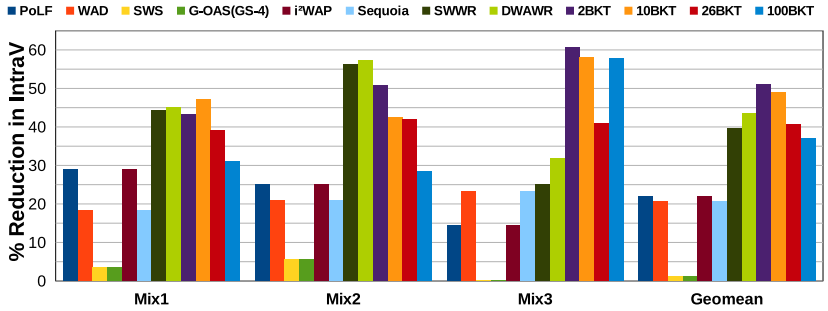

Figure 4.21: Percentage Reduction of IntraV in multi-core systems for *Hardware Efficient Approach*.

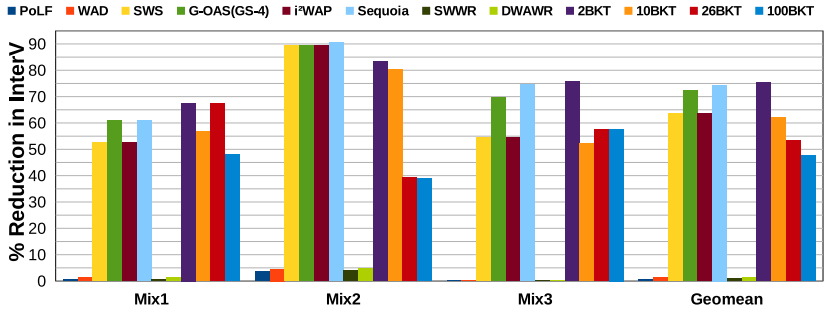

Figure 4.22: Percentage Reduction of InterV in multi-core systems for *Hardware Efficient Approach*.

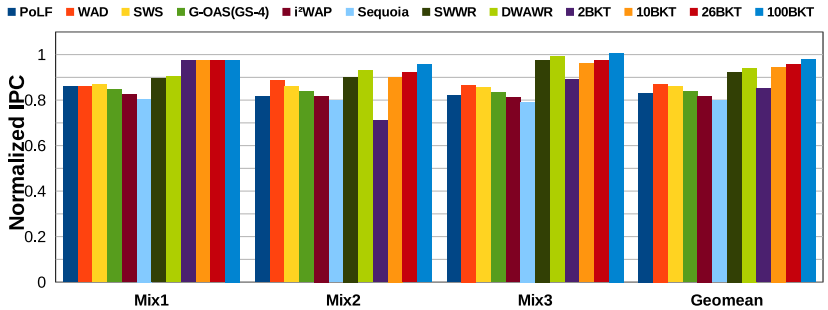

Figure 4.23: Normalized IPC in multi-core systems for *Hardware Efficient Approach*.

of the Primal Approach compared to the state-of-the-art techniques. It shows that how much more effective our Primal Approach is in improving the Lifetime of the STT-RAM

based LLC with minimal impact on the performance.

Table 4.9: Geomean Comparisons for Multi-core Systems with *Hardware Efficient Approach*.

Technique	RLI	% red IntraV	% red InterV	NIPC
PoLF	10.47	21.96	0.7	0.83
WAD	10.31	20.7	1.17	0.87
SWS	15.93	0.93	63.56	0.86
G-OAS(GS-4)	14.79	1.08	72.39	0.83
i^{2}WAP	21.37	21.96	63.5	0.81
Sequoia	22.49	20.7	74.42	0.79
SWWR	8.09	39.71	0.83	0.92
DWAWR	19.49	43.41	1.27	0.94
2BKT	32.54	51.09	75.20	0.84
10BKT	27.66	48.83	61.96	0.94
26BKT	24.36	40.61	53.47	0.95
100BKT	21.77	37.05	47.62	0.97

4.5.3 Sensitivity Analysis

4.5.3.1 Primal Approach vs Hardware Efficient Approach

All comparisons done in this section is with single core workloads of SPEC2006 and GAP workloads. Figure 4.24 shows the trends in the reduction of IntraV and InterV in a single-core system. It depicts that the reduction in IntraV and InterV increases with an increase in the SWAP % in the case of the *Primal Approach*. In the case of *Hardware Efficient Approach* both the IntraV and InterV decrease with an increase in bucket count. In this section, all the other Figures 4.25, 4.26 and 4.27 depict the various configurations of both: (a) *Primal Approach* presented in blue and (b) *Hardware Efficient Approach* presented in red. In Figure 4.25 it can be observed that the lower the bucket count, the larger the bucket size, therefore 2BKT configuration will have more number of write redirection than the 100BKT configuration for the *Hardware Efficient Approach*. In the case of the Primal Approach, it can be determined that the larger percentages of SWAP has more redirection as more write hot blocks are selected for block swapping. Figure 4.26 compares the effect of the different configurations on the Relative Lifetime Improvement of the two Approaches. It can be observed that the general trend of increase in RLI with an increase in SWAP percentage in *Primal Approach*, while the RLI decreases with an increase in the bucket count for *Hardware Efficient Approach*.

One observation we can see that in the case of 2BKT in *Hardware Efficient Approach* even though it selects 50% of the cache for redirection having higher redirection count it has lower RLI, lower reduction in InterV and IntraV than 30%SWAP. This is because in 2BKT the redirection is not Primal but random, i.e., the wear leveling is not being targeted always at the specific locations, whereas in 30%SWAP writes are redirected from write hot blocks to cold blocks. Figure 4.27 highlights that random redirection in the

Hardware Efficient Approach can lead to a significant reduction in performance even more than the *Primal Approach*.

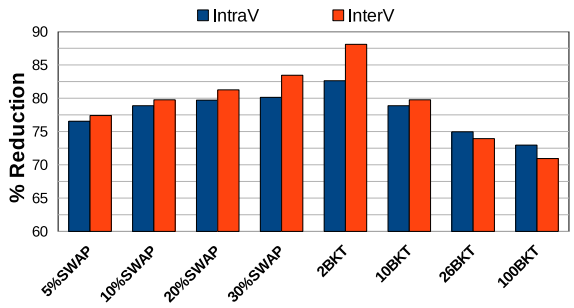

Figure 4.24: Comparison of Reduction in IntraV and InterV in a Single-core System.

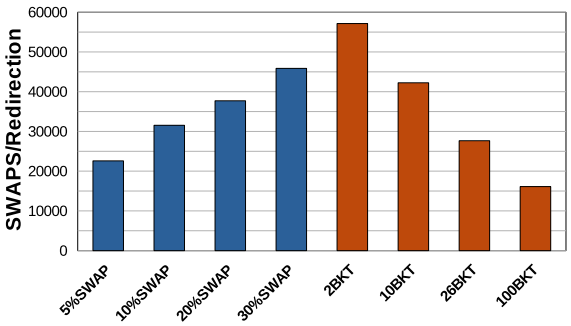

Figure 4.25: Comparison of Redirection/SWAP Count in a Single-core System.

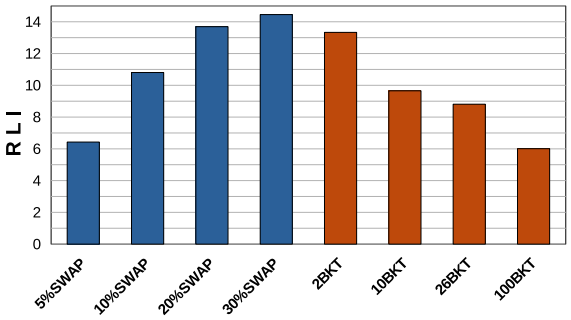

Figure 4.26: Comparison of Relative Lifetime Improvement in a Single-core System.

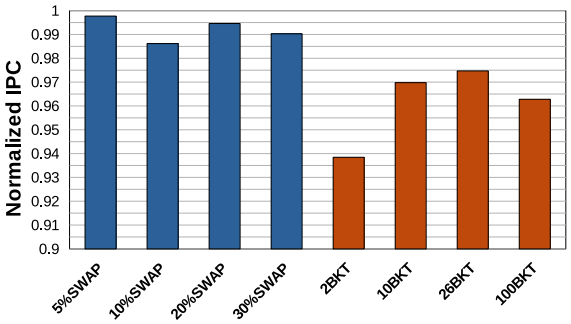

Figure 4.27: Comparison of Normalized IPC in a Single-core System.

Table 4.10 provides insights into how our techniques behave with increasing LLC size. All values are the geometric mean of the write-intensive workloads in a single-core system. Some observable trends from the table are:

- The reduction in IntraV and InterV decreases with an increase in the LLC cache size. This occurs because the larger the cache size the more the IntraV and InterV as larger cache sizes have a greater chance of encountering complex write patterns which are not always solvable by the *Primal Approach*.
- RLI increases with the increase in cache size. This occurs because the larger the cache size the more distributed the writes will be and the lower the max write count.
- The number of SWAPS/redirection decreases with the increasing cache size because the number of incoming write-backs to the buckets is also distributed.
- The NIPC values of the proposed approaches show higher degradation in *Hardware Efficient Approach* due to random redirection within a whole bucket. In case of 2BKT it can land in any position in half of the cache.

Table 4.10: Comparisons of various configurations of the proposed technique with varying cache sizes.

Configuration	LLC Size	% Reduction in IntraV	% Reduction in InterV	Relative LI	Normalized IPC	No. of SWAPS/ Redirection
5%SWAP	4MB	76.55	77.40	6.43	0.9977	22593.94
10%SWAP	4MB	78.88	79.76	10.81	0.9862	31549.39
20%SWAP	4MB	79.71	81.26	13.69	0.9946	37716.00
30%SWAP	4MB	80.13	83.45	14.45	0.9903	45868.52
2BKT	4MB	82.63	88.09	13.33	0.9385	57141.94
10BKT	4MB	78.88	79.76	9.65	0.9698	42242.31
26BKT	4MB	74.96	73.94	8.81	0.9747	27662.96
100BKT	4MB	73.20	72.20	6.01	0.9628	16115.26
5%SWAP	8MB	35.23	67.23	11.18	0.8745	7563.59
10%SWAP	8MB	40.02	73.29	13.21	0.8307	9346.62
20%SWAP	8MB	45.18	76.98	14.29	0.7891	11530.69
30%SWAP	8MB	48.02	85.11	15.50	0.7409	12928.29
2BKT	8MB	47.00	76.40	16.73	0.7292	16939.49
10BKT	8MB	42.72	72.05	13.66	0.7765	9346.62
26BKT	8MB	39.30	66.72	10.47	0.8268	5771.59
100BKT	8MB	37.87	58.11	7.91	0.8804	1797.31
5%SWAP	16MB	25.09	54.61	11.52	0.9005	2355.14
10%SWAP	16MB	28.51	59.54	16.65	0.8994	4288.09
20%SWAP	16MB	32.18	62.53	19.26	0.8549	9356.41
30%SWAP	16MB	34.21	69.14	24.99	0.7687	12930.85
2BKT	16MB	33.48	62.06	22.47	0.7125	17871.05
10BKT	16MB	30.43	58.53	14.70	0.8125	4288.09
26BKT	16MB	28.00	54.20	11.52	0.8592	2267.80
100BKT	16MB	26.98	47.21	8.31	0.9077	1038.63

4.5.3.2 SPEC2017 vs SPEC2006 vs GAP

This section presents a comparative analysis of benchmark outputs obtained from our proposed techniques, the primal approach and the hardware-efficient approach, under various configuration settings. The results are based on single-core systems executing write-intensive workloads, with the geometric mean values used for graphical representation.

Figure 4.28 illustrates the normalized IPC variations across different workloads. The observed trends reveal strong similarities between SPEC2017 and SPEC2006 benchmarks, while the GAP benchmarks, which involve more intensive write operations, exhibit higher normalized IPC values than their SPEC counterparts. The maximum geomean normalized IPC values recorded were 0.96 for SPEC2017, 0.97 for SPEC2006, and 1.07 for GAP benchmarks, demonstrating the advantage of our proposed techniques in handling write-heavy workloads.

Figure 4.29 provides a detailed comparison of percentage reduction in IntraV across different workloads when executed on our proposed architectures. While SPEC2006 and SPEC2017 follow similar trends, SPEC2017 exhibits more pronounced variations across different configurations. In contrast, the GAP benchmarks display significantly higher reductions in IntraV, a direct result of their greater write intensity compared to SPEC benchmarks.

Figure 4.30 shows the percentage reduction in InterV across all three benchmark categories. The results indicate that SPEC2017, SPEC2006, and GAP benchmarks follow similar trends, with the GAP benchmarks exhibiting a more pronounced reduction. Notably, SPEC2006 achieves a greater reduction in InterV than SPEC2017, underscoring the efficiency of our proposed techniques in mitigating InterV across different workloads.

Figure 4.31 presents the relative lifetime improvement of various configurations compared to the baseline LRU configuration. Once again, SPEC2006 and SPEC2017 display closely matching trends and values, whereas the GAP benchmarks exhibit significantly higher RLI values, reinforcing the advantages of our techniques in prolonging memory lifetime under write-intensive conditions.

Based on these findings, we conclude that our proposed techniques demonstrate superior performance on highly write-intensive GAP benchmarks compared to SPEC benchmarks. The heightened impact of our optimizations on GAP workloads confirms that our approach is particularly effective for environments with frequent write operations, further validating its role in enhancing system performance and memory longevity.

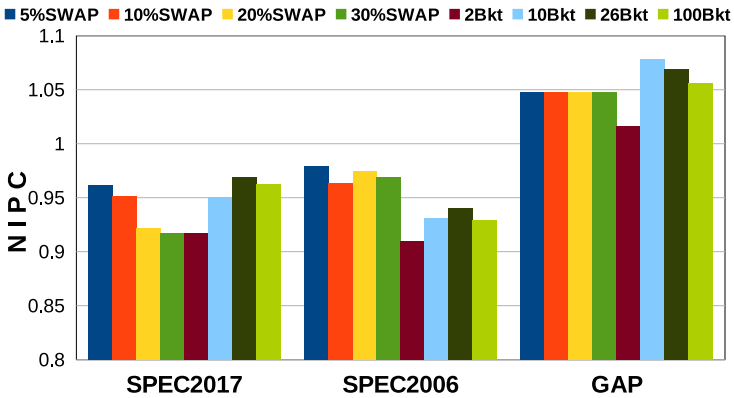

Figure 4.28: Normalized IPC of various different benchmarks run on various configurations of our proposed approaches.

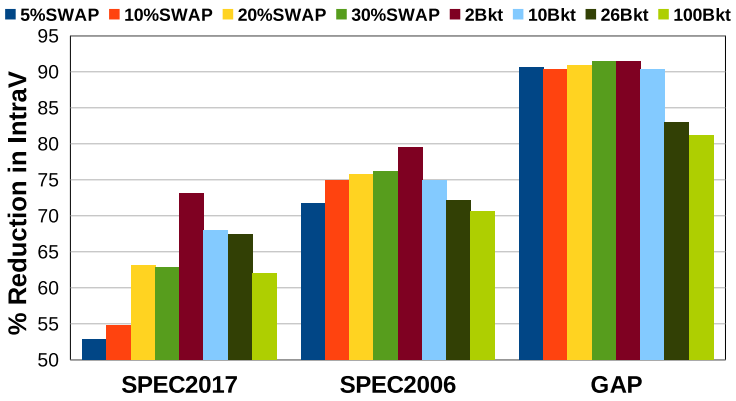

Figure 4.29: Reduction in IntraV of various different benchmarks run on various configurations of our proposed approaches.

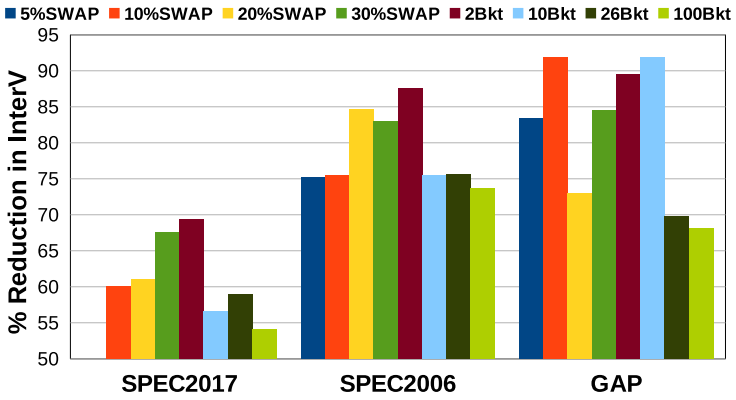

Figure 4.30: Reduction in InterV of various different benchmarks run on various configurations of our proposed approaches.

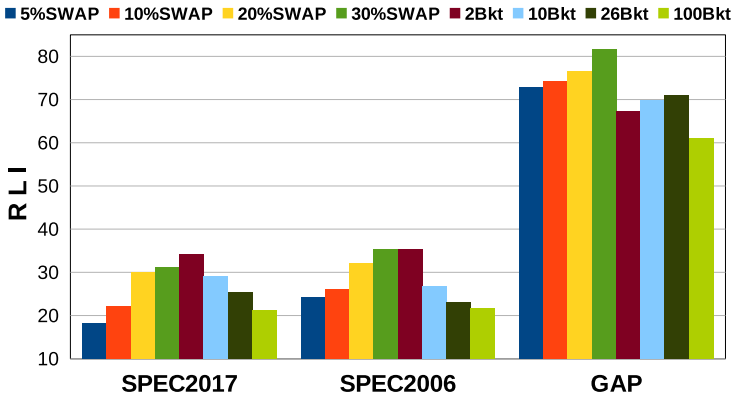

Figure 4.31: RLI of various different benchmarks run on various configurations of our proposed approaches.

4.5.3.3 Analysis of AI Workloads

AI workloads are write intensive and can lead to a large number of writes into the LLC. In order to show how AI workloads behave we have conducted an analysis of a few AI workloads like deepsjeng, leela, and exchange2 for the various configurations of the Primal

and Hardware Efficient Approach. Figure 4.32 depicts the RLI brought about by the various configurations on AI workloads. It can be clearly seen that the Primal Approach with increasing SWAP% is more effective than the Hardware Efficient Approach which decreases in efficiency with the increase in the number of buckets.

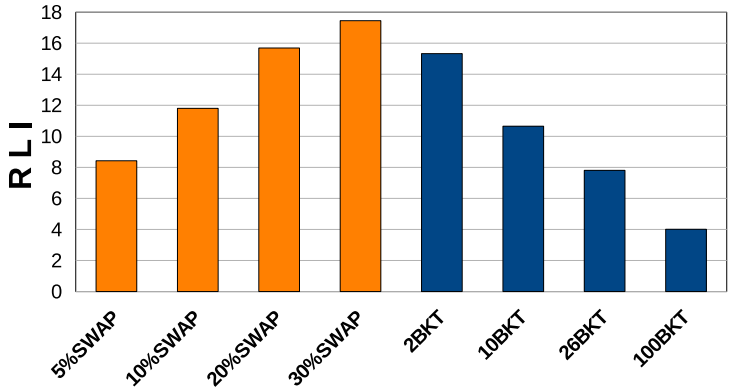

Figure 4.32: RLI of AI workloads on various configurations.

Figure 4.33 shows the varying degree of reduction in the InterV and IntraV when AI workloads are run with various configurations of the Primal and Hardware Efficient Approach. It shows that 30%SWAP is the most effective in reducing both the InterV and IntraV in case of Primal Approach while 2BKT having least number of buckets is most effective in case of Hardware Efficient Approach. The reduction in WV increases with an increase in the SWAP% while it decreases with the increase in the number of buckets.

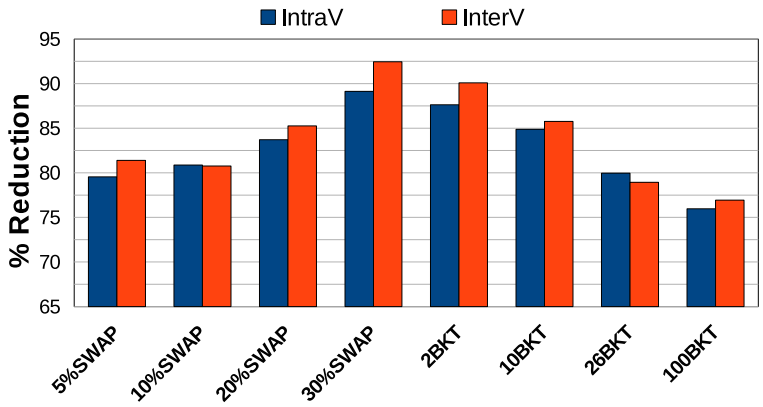

Figure 4.33: % Reduction of InterV and IntraV of AI workloads on various configurations.

4.5.4 Overhead Analysis

In the case of the Primal Approach, we employ an extra 8-bit write counter per block in the data array. Along with that, we have an additional valid bit and a $\log_2$B bit backward pointer. There is also a $\log_2$B bit forward pointer and a valid bit for every SRAM tag array entry. 'B' denotes the total number of blocks in the cache. In case a write counter is saturated after a certain epoch then we will half that specific write counter of the data

array block. In case of *Hardware Efficient Approach* we try to minimize the number of write counters and employ only one single counter per bucket. Different bucket sizes have different write counter sizes as the larger the bucket we have to accommodate larger counter size. We employ a FPTR and BPTR also for all the configurations of *Hardware Efficient Approach*, and the size of the FPTR and BPTR are similar to the ones in the *Primal Approach*.

Table 4.11 provides details of the overall storage overhead required with various configurations of the proposed approaches as compared with other state-of-the-art techniques. Table 4.11 details how different sizes of the LLC can change the size of the FPTR and BPTR. It also details how the bucket counter size in the case of *Hardware Efficient Approach* varies with the changing LLC size. Here, the value of P & S are determined as:

$$P = \lceil \log_2(\text{BS}) \rceil \tag{4.1}$$

$$S = \lceil \log_2 \left(\frac{10^9}{\text{BS}} \right) \rceil \tag{4.2}$$

P is indicative of the number of bits required to map all the blocks into a bucket in the *Hardware Efficient Approach* and S denotes the varying write counter size of the *Hardware Efficient Approach*. The term *BS* is the number of bits required to represent the Bucket Size. It is used to compute 'E' which the total size of the write counters.

Table 4.11: Storage Overhead of Proposed Techniques.

Configuration	LLC Size A	No. of Blocks B=A/64Bytes	FPTR&BPTR C=B*$\log_2$B (bits)	Bucket ID D=BS*P (bits)	Total WCTR E=B*8 / E=BS*S (bits)	Valid Bit F=B*1bit (bits)	Total Overhead (2*C+D+E+F) (KB)	% Overhead over baseline
Q%SWAP	4MB	65536	1048576	0	524288	65536	328	8
2BKT	4MB	65536	1048576	2	60	65536	264.007	6.4
10BKT	4MB	65536	1048576	40	270	65536	264.03	6.446
26BKT	4MB	65536	1048576	130	676	65536	264.09	6.447
100BKT	4MB	65536	1048576	700	2400	65536	264.378	6.45
Q%SWAP	8MB	131072	2228224	0	1048576	131072	688	8.39
2BKT	8MB	131072	2228224	2	60	131072	560	6.83
10BKT	8MB	131072	2228224	40	270	131072	560.037	6.836
26BKT	8MB	131072	2228224	130	676	131072	560.098	6.837
100BKT	8MB	131072	2228224	700	2400	131072	560.37	6.84
Q%SWAP	16MB	262144	4718592	0	2097152	262144	1440	8.789
2BKT	16MB	262144	4718592	2	60	262144	1184.007	7.22
10BKT	16MB	262144	4718592	40	270	262144	1184.0378	7.226
26BKT	16MB	262144	4718592	130	676	262144	1184.0983	7.227
100BKT	16MB	262144	4718592	700	2400	262144	76.0473	7.2288

Table 4.11 offers a detailed analysis of the storage/hardware overhead of our techniques as the LLC size increases. The reported values are for different configurations of our proposed approaches in a single-core system. Key trends observed in the table include:

- In case of *Primal Approach* the hardware overhead is the same for all configurations having the same LLC size.
- In case of *Hardware Efficient Approach* we can clearly see a trend that with the increase in the number of buckets the storage overhead increases. This is because with the increase in the number of buckets, it employs more counters per buckets which takes up more storage space.

- In the case of *Primal Approach* we see that with increasing LLC size the overall storage overhead was also increasing sizably, as each block needed to have its own write counter.
- In case of *Hardware Efficient Approach* the increment in the total % overhead is minimal and gradual as compared to *Primal Approach*, which occurs mainly due to the increase in the number of valid bits and the bits required for addressing the FPTR and BPTR.

Another overhead that exists is due to the extra energy required for the `SWAP` operations in Primal Approach. Each `SWAP` operation always entails an extra write. The redirection operation does not incur any extra writes and therefore it does not have any energy overhead. As we can see the increase in `SWAP` operations with the increase in aggressiveness of swapping. If we consider a very high retention time STT-RAM based LLC then the write energy per STT-RAM cell is 1.916nJ [113]. Therefore, 30%SWAP configuration in a 4MB shared cache will have an energy overhead of nearly 89902.29nJ per 1 Billion instructions, which is equivalent to a 4.9% energy overhead over the baseline LRU configuration.

4.6 Conclusion

Through our research, we have emphasized the simple modifications that are needed for adopting STT-RAM LLCs as a superior alternative to SRAM-based LLCs. Although STT-RAM offers significant advantages, a full-scale implementation of STT-RAM LLCs is not without its challenges. Issues such as potential performance degradation and concerns about the technology's limited lifetime must be carefully considered and addressed to fully realize the benefits of this approach. Therefore, we introduce SmartDeCoup, an innovative architecture that is not only power-efficient and cost-effective but also strategically engineered to address the challenges associated with STT-RAM LLCs. SmartDeCoup aims to significantly reduce IntraV and InterV, which are key factors in the wear and tear of STT-RAM cells, thereby dramatically improving the overall lifetime of the STT-RAM LLC. Importantly, this architecture achieves these enhancements without sacrificing much system performance, ensuring that the benefits of STT-RAM technology can be fully realized in a reliable and sustainable manner. This makes SmartDeCoup a compelling solution for those looking to optimize the trade-offs between performance, longevity, and cost in next-generation computing systems while providing a framework for security setups.

Chapter 5

Endurance Attacks on STTRAM LLCs

Malicious attacks in a multi-core setup need access to only a single core to perform repeated attacks on specific memory locations that can lead to an accelerated lifetime degradation of the STT-RAM LLC cells. To highlight this vulnerability of STT-RAM LLC we propose four variations of TENDRA (Targeted Endurance Attack), namely, Recurring Location Attack (RLA), Recurring Toggle Attack (RTA), Random Location Attack (RnLA) and Random Toggle Attack (RnTA). Our work highlights the efficiency of these attacks on modern counter based wear leveling techniques and also the effect of wear leveling on these attacks.

Publications from this Chapter

- **Prabuddha Sinha**, Mangena Likhit Sai, Shirshendu Das and Venkata Tavva Kalyan, "TENDRA: Targeted Endurance Attack in STT-RAM based LLCs", in IEEE Embedded Systems Letters. [DOI: 10.1109/LES.2024.3502297], also part of the SeHAS Workshop at HiPEAC 2025.

5.1 Introduction

Uneven write distribution is a major factor aggravating the limited write endurance issue of STT-RAM LLC. Modern multi-core processors typically have a shared LLC with different private L1 and L2 caches. Write-backs from different applications from their respective L2's lead to diverse/complex write patterns at the LLC. Accordingly, in order to increase the lifetime of the STT-RAM LLC, various counter based wear leveling techniques were introduced [12, 13]. These wear leveling techniques try to remap the heavily written cache lines to other memory locations. This work stands-apart and primarily presents a detailed study of how existing counter wear leveling techniques are vulnerable to endurance attacks. Endurance attacks, as the name suggests, aim to increase the writes to the STT-RAM, leading to an accelerated wear-out of the cells. The accelerated wear-out occurs due to repeated writes to specific locations in the STT-RAM. Endurance attacks not only accelerate the degradation of STT-RAM, drastically shortening its lifetime, but also create significant security vulnerabilities, potentially compromising the integrity and reliability of the system.

5.2 Motivation

Most existing wear leveling techniques are proposed and tested only with standard benchmark applications that are benign in nature [112]. These techniques can be bypassed to launch an endurance attack. STT-RAM cells writes data by flipping the magnetic orientation of layers within a magnetic tunnel junction using spin-polarized currents. The high current densities required for writes lead to electron migration and degradation of the tunnel barrier over time. Over many write (or erase) cycles, the incremental damages accumulate, leading to increased error rates and eventually to cell failure. The physical limits of the materials used determine the maximum number of reliable sustainable write cycles, known as the write endurance limit. Non-uniform writes lead to accelerated degradation of specific targeted STT-RAM cells. Without wear leveling, this uneven wear can cause premature failure of certain regions of the memory, reducing the overall lifespan of the device. For example, user can deliberately run a malicious program to hasten the wear-out of the STT-RAM, particularly when the warranty period of the product is nearing its end, with the intention of receiving a replacement system from the manufacturer under the guise of legitimate failure. Consequently, the security challenges in STT-RAM based LLCs extend beyond threats posed by malicious attackers for manipulating or retrieving data to include opportunistic exploitation by users seeking to manipulate vulnerabilities for personal gain, such as acquiring replacement machines at will. Until practical and reliable secure wear-leveling algorithms are developed, it is improbable that industrial manufacturers will risk designing main memory systems with STT-RAM components. As LLC is also an on-chip memory any damage to the LLC will require the whole CPU to be swapped out as a whole, making the system more vulnerable to attack. Various attack techniques have been proposed for NVM based Main Memory specifically PCM based MM [114, 115, 116]. These attack techniques develop a metric called LVF (Line Vulnerability Factor) that show how vulnerable a line is after some attack on the system. They also propose mitigation techniques like Start-Gap Wear Leveling and Region Based Start-Gap Wear Leveling and its variations, that help in redistributing the writes evenly and reduce the write variation thereby improving the lifetime. Rathi et al. [117] also proposed various techniques to model attacks on NVM based LLCs and the challenges faced in implementing NVM based LLCs.

Addressing non-uniform writes, various wear leveling techniques have been proposed for STT-RAM LLC. The most common type of wear leveling techniques proposed are the counter based techniques, a few recent techniques being i^2WAP [12], SWWR, DWWR and DWAWR [14]. These techniques monitor the regions of the LLC experiencing the highest concentration of writes within a given time frame and mitigate wear by redirecting writes from these hotspots to less active areas of the cache. However, malicious programs can exploit these wear-leveling mechanisms if they acquire prior knowledge about the locations of write-intensive regions, the destinations of redirected writes, or the timing of redirection events. Armed with this insight, attackers can strategically time their writes

to target specific locations, bypassing these defenses and potentially accelerating cache wear. Furthermore, in wear leveling techniques writes need to be redirected in order to facilitate efficient wear leveling of the shared STT-RAM LLC, various malicious programs can take advantage of these algebraic based remapping as they are not deterministic [115] and are not remapped directly from a table.

5.3 TENDRA: Targeted Endurance Attack

5.3.1 Threat Model

This study explores the design and functionality of a multi-core architecture built around a sophisticated three-level cache hierarchy. Within this system, each core is provisioned with private SRAM-based L1 and L2 caches to ensure rapid access to frequently used data. Complementing this is a shared STT-RAM Last-Level Cache (LLC), which serves as an inclusive caching layer accessible by all cores. The research delves into a representative use case where a multi-core system operates in a shared environment, accommodating multiple users simultaneously. Each user executes tasks with distinct computational requirements, leading to a dynamic interplay of workloads. As these tasks share processing resources, the system must adapt to the diverse computational intensities and data access patterns exhibited by applications running across different cores. This complex environment provides a compelling context for examining the interplay between workload characteristics and cache performance in multi-core systems.

The wide variety of workload and data access patterns contributes to a uniquely diverse write distribution across the shared LLC, capturing the inherent heterogeneity of the system's operations. This intricate distribution significantly influences overall system performance while simultaneously unveiling potential areas of vulnerability that could be exploited or require optimization.

A malicious user could, for instance, exploit this architectural vulnerability through meticulously orchestrated endurance attacks. By deliberately engineering workloads to intensify stress on targeted regions of the cache, these attacks could hasten the degradation of the STT-RAM LLC, progressively diminishing system performance and potentially jeopardizing its long-term stability and reliability. This vulnerability underscores the importance of robust cache management and security mechanisms to ensure the reliability and integrity of such multi-core architectures. Any attacker can utilize their malicious endurance attack application and be a nuisance to all the other applications, by targeting writes specifically to a few predetermined regions in the LLC. Therefore, the attacker needs to just get access to one single core to run malicious application in order for it to be a threat to the whole system.

5.3.2 The Attack Idea

The four different TENDRA explored on top of existing wear leveling techniques, namely, Recurring Location Attack (RLA), Recurring Toggle Attack (RTA), Random Location Attack (RnLA) and Random Toggle Attack (RnTA) are explained next.

5.3.2.1 Recurring Location Attack (RLA)

RLA allocates a memory to an array (say `large-array`) that is larger than L2 cache (a pseudo-code is shown below) and keeps updating the entries of `large-array`. Since the array size is larger than the L2 capacity, the updates performed in `line 3` of the RLA result in write-backs from L2. Hence, for the internal loop (`line 2`), when the value of i reaches N, only the last part of the array remains in the L2. The remaining parts must be written-back to LLC as all the cache blocks must be dirty. When the whole process repeats multiple times (`line 1`), it causes a lot of write-backs from L2 to LLC. Since all the writes are happening to a fixed memory location allocated to the array, the writes target only fixed LLC locations. Due to such targeted writes in an LLC, a drastic reduction in the lifetime of the whole LLC is observed. If the data to be written into the L2 cache is not present in the LLC, an additional write to the LLC occurs as in this case, the required data block must first be fetched from the main memory to the LLC before it can be written into the L2 cache. Furthermore, the write latency of an STT-RAM cell is quite high, nearly 100 cycles as compared to the read latency of 20 cycles [118]. Therefore, excessive write-backs due to the malicious RLA can cause a drastic reduction in the performance of the system as the write operations keep blocking (leading to contention) the read operations from other applications running on different cores [13]. To examine the impacts of RLA, we employ a compact yet effective program designed to generate traces through a repeated L2 flushing strategy. This approach systematically produces recurring write-back patterns within the LLC, offering a controlled and consistent mechanism to observe and analyze the resulting effects.

```
// Allocate a memory for large_array.
// Size should be more than L2.
1. while(true){
2.   for(i=0 to N) //N is array size.
3.     large_array[i]+=1;
4. }
```

Since RLA only targets a particular LLC part to where the allocated array maps, counter based wear leveling techniques can identify these hot-spots and redirect the write-backs to other less written locations of LLC. The proposed attack can be partially handled by the existing endurance enhancement techniques like DWAWR. Therefore, in order to make the attack more effective against these counter based wear leveling techniques we introduce another variation of attack.

5.3.2.2 Recurring Toggle Attack (RTA)

RTA targets two distinct memory locations by orchestrating repeated write-backs from the L2 cache to the LLC. True to its name, RTA alternates these write-backs between the two locations, toggling back and forth. After writing to one location, the process randomly determines the delay before toggling back to the other, introducing unpredictability into the pattern. Notably, the timing of these toggles is entirely random, generated dynamically during trace creation, and remains beyond user control. This randomness adds complexity to the behavior and analysis of such memory interactions.

```
// Allocate a memory for two large_arrays.
// Size should be more than L2.
1. while(true){
   //Frist write on array1. Keep writing until randomly decided to stop.
2.        while(true){
3.              for(i=0 to N) //N is array size.
4.                  large_array1[i]+=1;
                    //randomly select true or false.
5.                  if (false) break;
6.                    }
    //Once the write on array1 stops, start writing on array2.
    // Writing on array 2 will also stop randomly.
7.        while(true){
8.              for(i=0 to N) //N is array size.
9.                    large_array2[i]+=1;
                      //randomly select true or false.
10.                   if (false) break;
11.                    }
  //The same process will continue infinitely.
12.}
```

The randomization of switching time is a key factor in RTA's effectiveness against wear leveling techniques. Traditional wear leveling mechanisms rely on fixed time intervals to monitor write activity in the Last-Level Cache (LLC) and identify hotspots. Once a region is flagged as a write hotspot, it is temporarily closed off to further writes, redirecting new write traffic to other parts of the cache to distribute wear more evenly.

RTA negates these counter-based wear leveling strategies by leveraging its random toggle behavior. Instead of persisting with writes to the previously identified hotspot which would trigger redirection by the wear leveling mechanism, RTA switches its repeated write activity to another target location. This new target is likely not flagged as a hotspot at that moment. Consequently, even as the wear leveling mechanism redirects writes from the previously flagged region, RTA has already shifted its focus to a new area.

The random timing of these toggles ensures that RTA's write patterns remain out of sync

with the fixed intervals used for hotspot detection. By the time wear leveling begins redistributing writes from a region previously determined as write-intensive, RTA has already moved its activity to a different location. As a result, the newly targeted region becomes the focus of concentrated write activity, while the previously mitigated region experiences minimal new writes. This strategic timing undermines the effectiveness of wear leveling, allowing RTA to sustain its attack without significant interruption.
To maximize the write count, we focus on targeting only two predetermined memory locations, with the attacks directed alternately between these two spots. This strategic approach ensures a concentrated and controlled assault on the system. Additionally, during the execution of the RTA, if any normal, non-malicious writes occur to the identified hotspots, they inadvertently mislead the wear leveling mechanisms. These writes, which do not pose any real threat, cause the wear leveling techniques to mistakenly perceive increased write activity in those areas, thereby triggering unnecessary remapping processes. As a result, the wear leveling mechanisms are forced to expend additional resources managing these non-harmful writes, creating an unnecessary overhead that could have been avoided had the writes been correctly ignored. This disruption amplifies the inefficiency of the wear leveling process, further compromising the system's ability to handle the attack effectively.

5.3.2.3 Random Location Attack (RnLA)

The Random Location Attack (RnLA) is aptly named for its strategy of targeting unpredictable locations within the LLC through repeated write operations. It selects different locations at random and switches between them dynamically, with intervals that are themselves unpredictable. This creates complete uncertainty regarding both the timing and the location of each write operation. The intrinsic randomness of RnLA makes it particularly disruptive to counter-based wear-leveling techniques, which typically depend on identifying consistent "write-hot" spots to mitigate the wear of write-limited STT-RAM cells. By constantly shifting its focus across various regions of the cache, RnLA effectively circumvents these counter-based defenses, remaining undetected and elusive. As a result, wear-leveling strategies that are designed to track and manage patterns of repeated writes to fixed locations are unable to cope with the dynamic and randomized nature of RnLA. This exposes a significant weakness in conventional wear-leveling algorithms, revealing their struggle to adapt to attacks characterized by such high levels of unpredictability in both location and timing. Repeated unpredictable write operations pose a significant threat to the overall system reliability, as they can severely impact the lifespan of the STT-RAM Last-Level Cache (LLC). The constant and unpredictable nature of these writes accelerates the wear of the STT-RAM cells, shortening their operational lifetime. Additionally, the continuous stream of write operations introduces performance issues. Specifically, the time required for each write to execute in an STT-RAM LLC can lead to congestion in the read/write queue. This congestion delays the servicing of critical read requests, ultimately resulting in slower system performance and reduced efficiency in data

retrieval.

```
// Allocate multiple large_arrays with differnt base addresses.
// Size of arrays should be more than L2.
1. while(true){
2.   switch(n){ //n is random number between 1 to n;
3.     case 1: for(i=0 to N) //N is array size.
4.                large_array1[i]+=1;
5.                break;
6.     case 2: for(i=0 to N) //N is array size.
7.                large_array2[i]+=1;
8.                break;
9.                  .
10.                 .
11.                 .
12.                 .
13.    case n: for(i=0 to N) //N is array size.
14.                large_arrayn[i]+=1;
15.                break;
16.   }
17. }
```

5.3.2.4 Random Toggle Attack (RnTA)

The Random Toggle Attack (RnTA), as its name suggests, is specifically designed to alternate write operations between two randomly selected memory locations within the Last-Level Cache (LLC). Unlike conventional attack methods, RnTA introduces a high degree of unpredictability by making both the selection of target locations and the timing of each switch entirely random. This randomness serves as a critical feature of the attack, enhancing its ability to evade detection and mitigation efforts.

A fundamental distinction between RnTA and the Random Location Attack (RnLA) lies in their operational behavior. While RnLA focuses its write operations on a single, randomly chosen memory location, RnTA dynamically toggles between two separate locations. This back-and-forth switching pattern significantly complicates the task of counter-based wear-leveling techniques, which are designed to monitor and mitigate excessive writes to specific regions by identifying so-called "write-hot" locations. By unpredictably alternating between two memory regions, RnTA effectively circumvents these defensive measures, making it exceedingly difficult for wear-leveling mechanisms to determine which locations should be flagged and protected.

This ability to evade counter-based defenses highlights a crucial vulnerability in existing security measures. Since wear-leveling systems rely on tracking repeated access patterns to detect anomalies, an attack like RnTA, which disrupts conventional tracking methodologies

through random toggling, can successfully exploit this weakness. By continuously shifting its focus in a randomized manner, RnTA not only undermines conventional detection strategies but also exposes the limitations of current memory protection frameworks, making it a potent and stealthy attack vector.

```
// Allocate multiple large_arrays with differnt base addresses.
// Size of arrays should be more than L2.
1. while(true){
2.   switch(n){ //n is random number between 1 to n;
3.     case 1: for(i=0 to N) //N is array size.
4.                 large_array1[i]+=1;
5.                 large_array2[i]+=1;
                   //randomly select true or false.
6.                  if (false) break;
7.     case 2: for(i=0 to N) //N is array size.
8.                 large_array3[i]+=1;
9.                 large_array4[i]+=1;
                   //randomly select true or false.
10.                if (false) break;
11.                      .
12.                      .
13.    case n: for(i=0 to N) //N is array size.
14.                large_arrayn-1[i]+=1;
15.                large_arrayn[i]+=1;
                   //randomly select true or false.
16.                if (false) break;
17.   }
18.}
```

5.3.3 Comparison between Attacks

Table 5.1 provides a succinct overview of the four distinct targeted endurance attacks proposed, emphasizing the differences in their target selection and the specific techniques they are designed to exploit.

Table 5.1: Different TENDRA Comparisons.

Attack	Type	Target	Effectiveness
RLA	Endurance	Fixed	LRU
RTA	Endurance	Fixed Toggle	LRU + WL Techniques
RnLA	Endurance	Random	LRU + WL Techniques
RnTA	Endurance	Random Toggle	LRU + WL Techniques

The figure demonstrates that all the described attacks are designed with the primary objective of reducing the endurance of an STT-RAM-based Last-Level Cache (LLC). Each

attack type leverages a distinct strategy to exploit vulnerabilities in cache management and wear-leveling mechanisms:

- RLA employs a fixed targeting strategy, focusing on a single memory location. It is particularly effective against generic Least Recently Used (LRU) replacement policies.
- RTA alternates writes between two fixed memory locations, making it effective against both generic LRU policies and wear-leveling techniques.
- RnLA introduces randomness by dynamically selecting attack locations at unpredictable intervals. This randomization makes it highly effective against wear-leveling mechanisms, which struggle to adapt to its unpredictability.
- RnTA combines randomness with toggling behavior by alternating writes between two randomly selected sets of memory locations. This approach further amplifies its effectiveness against wear-leveling techniques, leveraging both location and timing unpredictability to evade countermeasures.

The attacks collectively highlight critical weaknesses in existing endurance and wear-management strategies, showcasing the need for more adaptive and robust countermeasures.

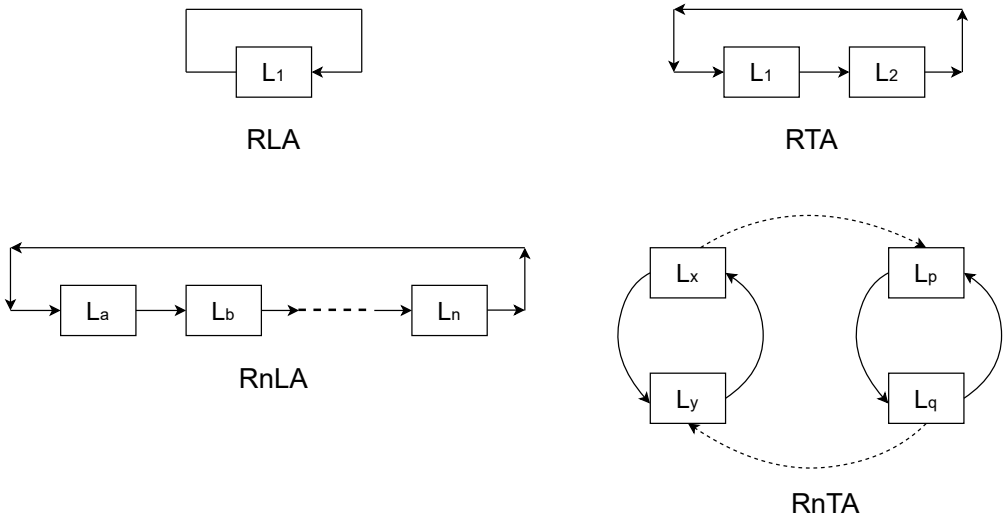

Figure 5.1: Visualization of the Different Types of Targeted Endurance Attacks.

Figure 5.1 provides a comprehensive visualization of how repetitive writes target various addresses within the STT-RAM LLC, showcasing the distinct attack patterns and their intensity across cache regions. It illustrates the specific behaviors of different attacks:

- RLA focuses on a single memory location, repeatedly directing writes to the same address.

- RTA alternates between two fixed memory locations, toggling write operations back and forth.
- RnLA employs a randomized approach, dynamically selecting different locations (denoted as L_x) to attack at unpredictable intervals.
- RnTA demonstrates a more complex behavior, involving two random sets of memory locations. Within each set, two locations are randomly chosen, and writes toggle between them. The attack then switches to another random set after an unpredictable duration.

This figure highlights the strategic variation in targeting methods used by these attacks, illustrating how their unpredictability and toggling behaviors challenge conventional cache management and wear-leveling mechanisms.

5.4 Experimental Evaluation

5.4.1 Simulator Setup

We use the trace-based ChampSim Simulator [97] to conduct experiments. Our "baseline" (LRU) is a configuration where no wear leveling technique is implemented as shown in Table 5.2. To understand the interaction of the proposed attacks and the wear leveling, we consider i^{2}WAP, SWWR and DWAWR techniques.

Table 5.2: Simulation parameters of the baseline multi-core system.

System Components	Parameters
Core	Out-of-order, bimodal branch predictor, 4 GHz with 6-issue width, 4-retire width, 352-entry ROB
L1I	32 KB, 8-way, 4 cycles
L1D	32 KB, 8-way, 5 cycles
L2	2MB 8-way associative, 10 cycles, LRU
LLC	32MB Shared STT-RAM, 16-way, RL: 20 cycles, WL: 100 cycles, LRU
MSHRs	8/16/32 at L1I/L1D/L2, 64/core at the LLC
DRAM controller	64-entry RQ and WQ, reads prioritized over writes, Burst write: 6/8th of queue size
DRAM chip	4KB row-buffer per bank, open page, burst length 16, t_{RP}: 12.5ns, t_{RCD}: 12.5ns, t_{CAS}: 12.5ns

5.4.2 Workloads

For the experiments, we use the write-intensive benchmarks from SPEC CPU 2017 benchmark suite [112] along with the proposed attacks in a four-core setup with each core executing 250M instructions after a warm-up of 50M. Experiments with other workloads follow similar trends as the mix of benchmarks provided in Table 5.3 hence we use them

alone. The attack traces were generated by a Pin tool [119] based tracer that is part of ChampSim. Each instruction in the trace is of the 64B size.

Table 5.3: Workloads for Quad-Core System.

Mix Type	Workloads
Mix0	mcf, lbm, xalancbmk and wrf
Mix1	RLA, mcf, lbm and xalancbmk
Mix2	RTA, mcf, lbm and xalancbmk
Mix3	RLA , RTA, mcf and lbm
Mix4	RnLA, mcf, lbm and xalancbmk
Mix5	RnTA, mcf, lbm and xalancbmk
Mix6	RnLA, RnTA, mcf and lbm
Mix7	RLA, RTA, RnLA and RnTA

5.5 Results and Analysis

5.5.1 Effects of RLA, RTA, RnLA and RnTA on different State-of-the-art Wear Leveling Techniques

Figures 5.2 through 5.65 illustrate the write count distribution, represented as heat maps, across the LLC for the workload mixes listed in Table 5.3 which include benign mixes like Mix0 and various combinations of our proposed attack traces. These are run on a quad core system. These mixes are evaluated using eight different techniques among them are the state-of-the-art: baseline (LRU), i^{2}WAP, SWWR, and DWAWR. We also evaluate against our four different proposed wear leveling techniques: PROLONG and LiveWay proposed in Chapter 3 along with Primal Approach and Hardware Efficient Approach proposed in Chapter 4. In order to emphasize the variability in write patterns and to highlight the increased write activity caused by attacks, the heat maps are scaled differently.

Figure 5.2 focuses on the write counts for LRU-Mix0, representing the baseline STT-RAM LLC configuration where only benign, write-intensive applications are present. The figure clearly reveals a lack of uniformity in write distribution within this baseline setup, underscoring inherent imbalances in the system. The observed non-uniformity in write distribution becomes significantly more pronounced under the baseline configuration when executed with RLA, as shown in Figure 5.10 (LRU-Mix1). In contrast, Figure 5.3 demonstrates that i^{2}WAP achieves a more uniform distribution of writes across the LLC under benign conditions. However, this uniformity is disrupted when RLA is introduced, as evident in Figure 5.11 (i^{2}WAP-Mix1). The concentrated writes in specific regions effectively negate the wear-leveling advantages initially provided by i^{2}WAP.

A similar pattern emerges when considering the SWWR and DWAWR techniques under RLA conditions, as shown in Figures 5.12 (SWWR-Mix1) and 5.13 (DWAWR-Mix1). Although these wear-leveling techniques exhibit slight improvements in write distribution, their effectiveness in achieving a uniform pattern under RLA remains limited. In contrast, the proposed techniques like PROLONG and LiveWay in Figures 5.14 (PROLONG-Mix1) and 5.15 (LiveWay-Mix1) show how effective these techniques are in evenly distributing

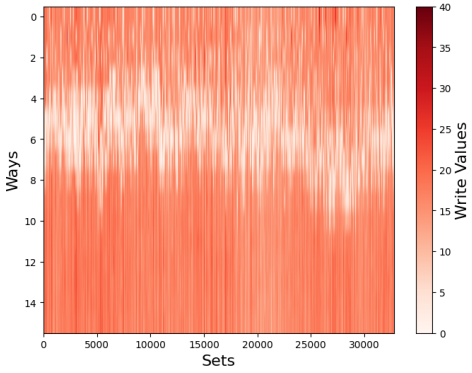

Figure 5.2: LRU-Mix0.

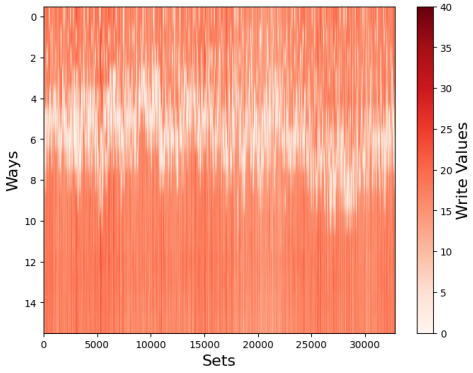

Figure 5.3: i^{2}WAP-Mix0.

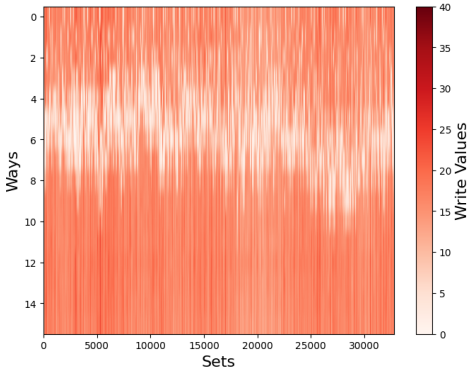

Figure 5.4: SWWR-Mix0.

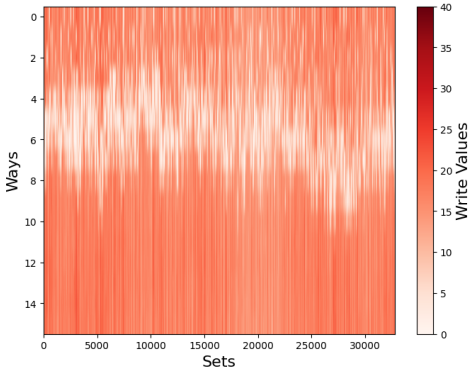

Figure 5.5: DWAWR-Mix0.

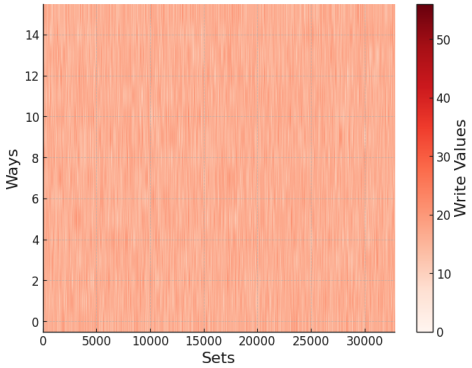

Figure 5.6: PROLONG-Mix0.

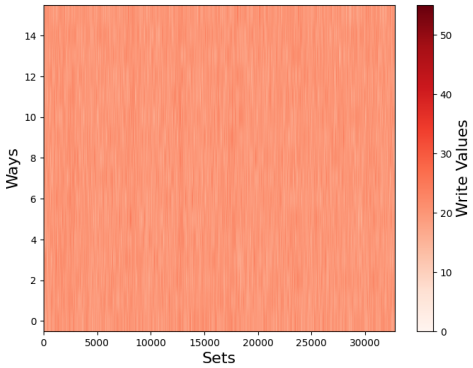

Figure 5.7: LiveWay-Mix0.

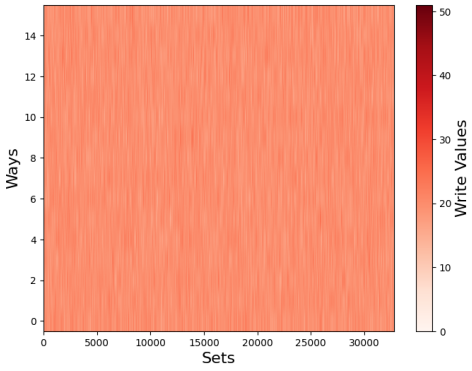

Figure 5.8: SDC:Primal-Mix0.

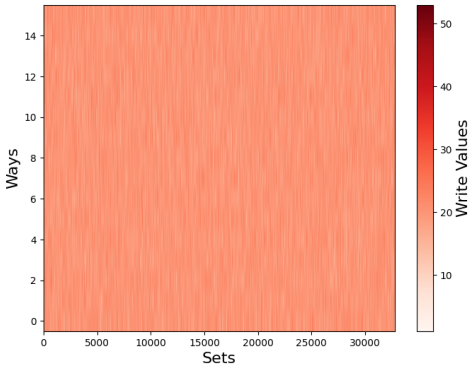

Figure 5.9: SDC:HE-Mix0

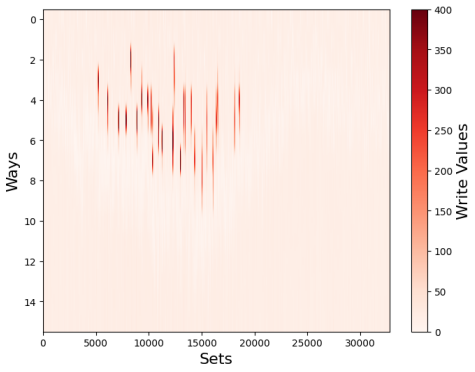

Figure 5.10: LRU-Mix1.

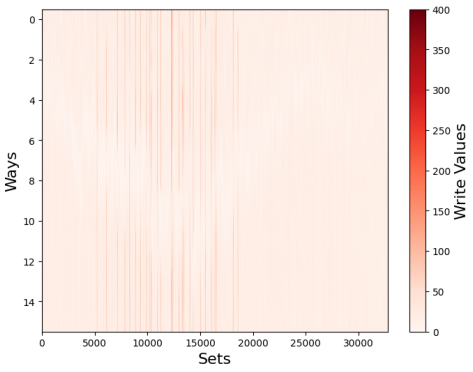

Figure 5.11: i^{2}WAP-Mix1.

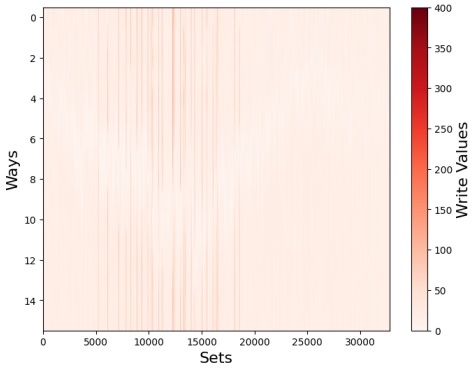

Figure 5.12: SWWR-Mix1.

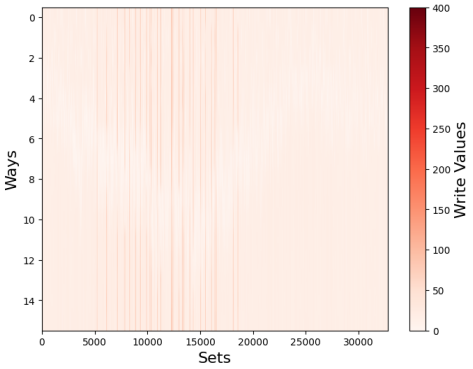

Figure 5.13: DWAWR-Mix1.

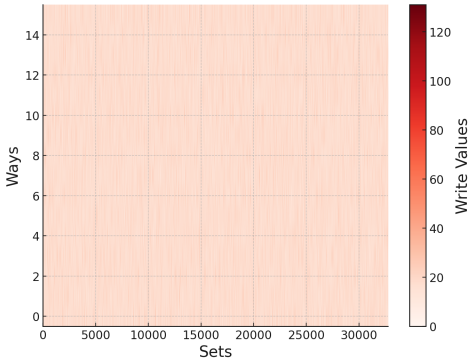

Figure 5.14: PROLONG-Mix1.

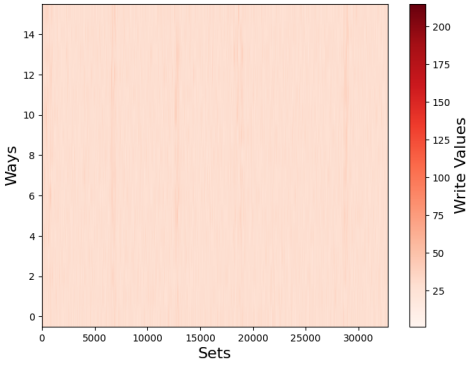

Figure 5.15: LiveWay-Mix1.

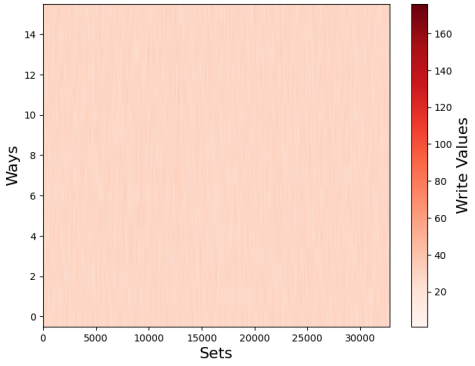

Figure 5.16: SDC:Primal-Mix1.

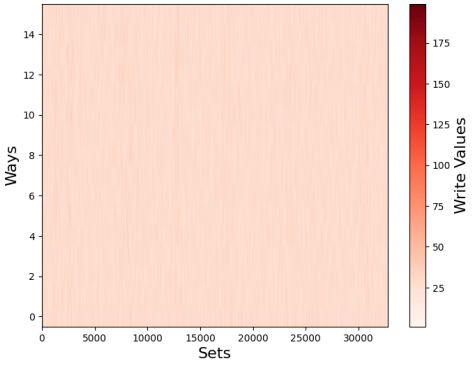

Figure 5.17: SDC:HE-Mix1.

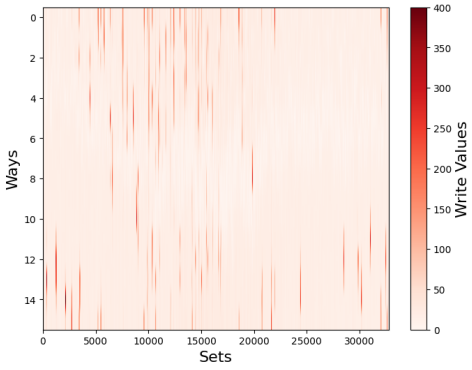

Figure 5.18: LRU-Mix2.

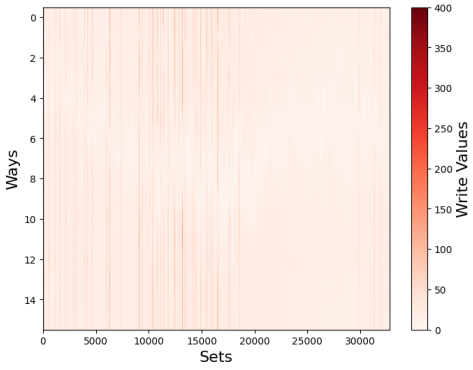

Figure 5.19: i^{2}WAP-Mix2.

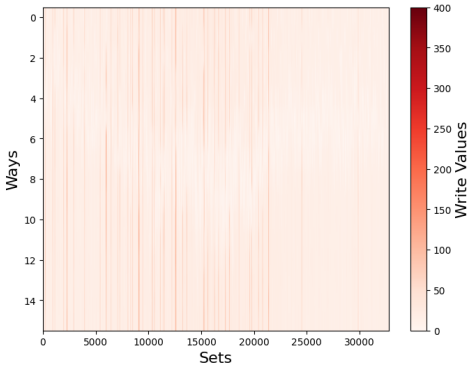

Figure 5.20: SWWR-Mix2.

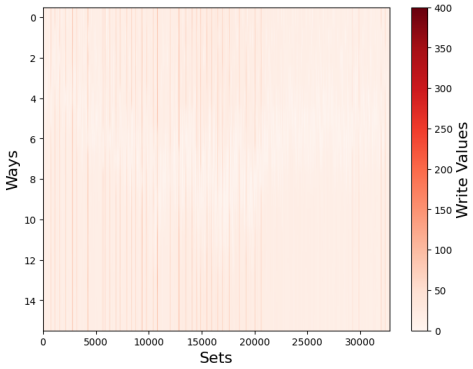

Figure 5.21: DWAWR-Mix2.

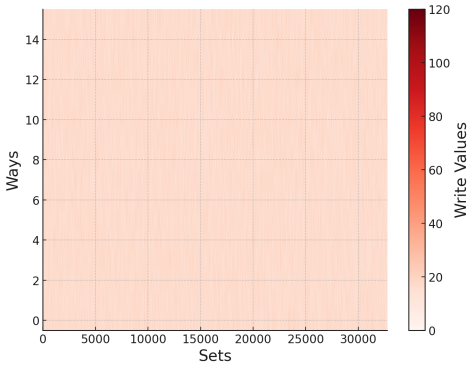

Figure 5.22: PROLONG-Mix2.

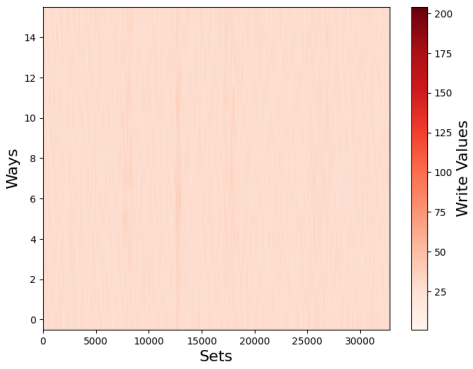

Figure 5.23: LiveWay-Mix2.

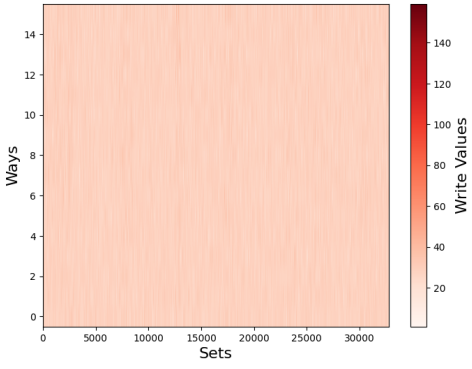

Figure 5.24: SDC:Primal-Mix2.

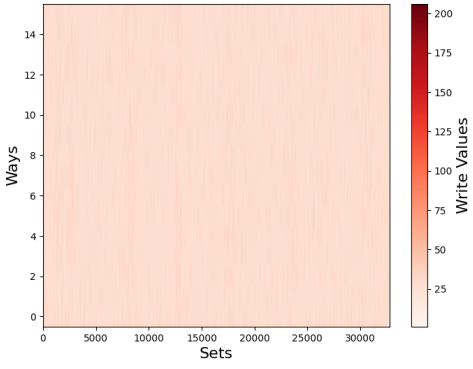

Figure 5.25: SDC:HE-Mix2.

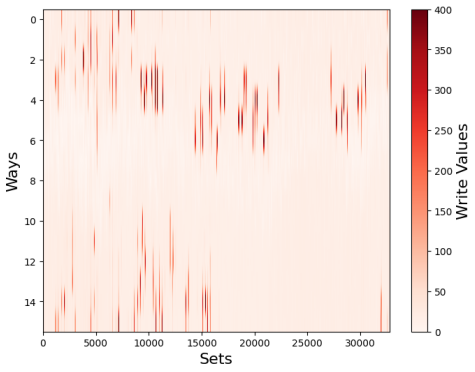

Figure 5.26: LRU-Mix3.

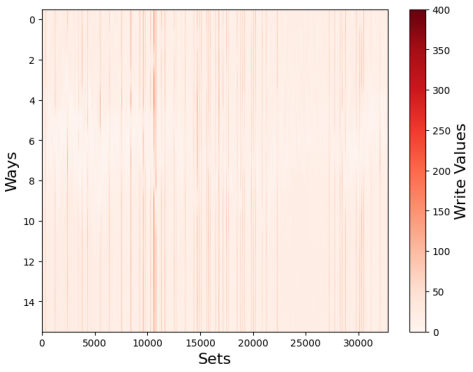

Figure 5.27: i^{2}WAP-Mix3.

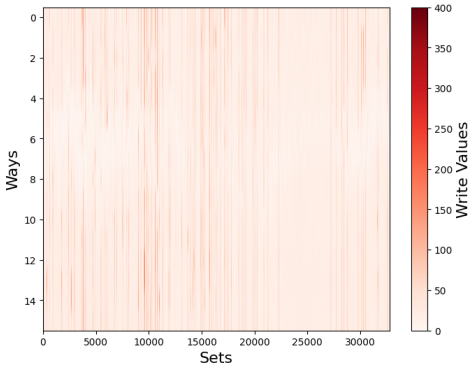

Figure 5.28: SWWR-Mix3.

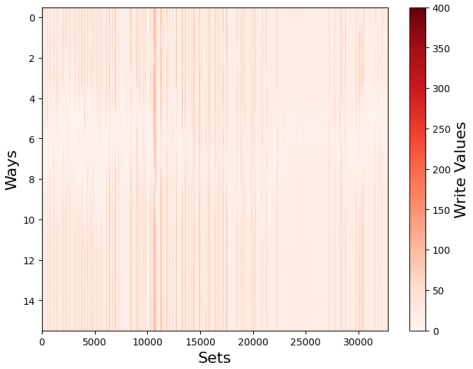

Figure 5.29: DWAWR-Mix3.

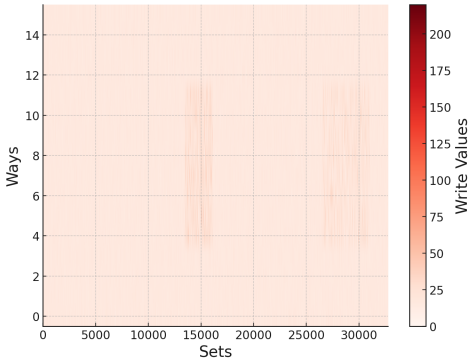

Figure 5.30: PROLONG-Mix3.

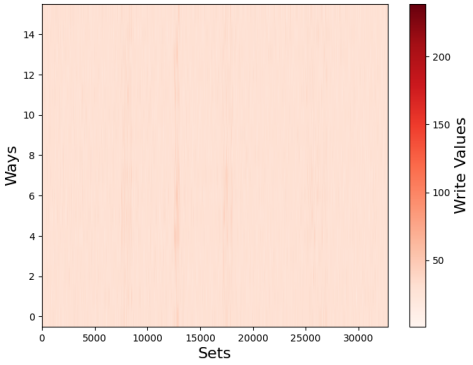

Figure 5.31: LiveWay-Mix3.

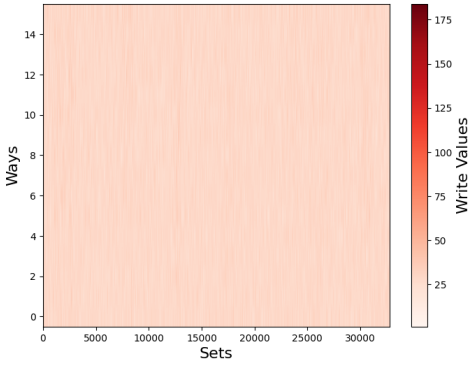

Figure 5.32: SDC:Primal-Mix3.

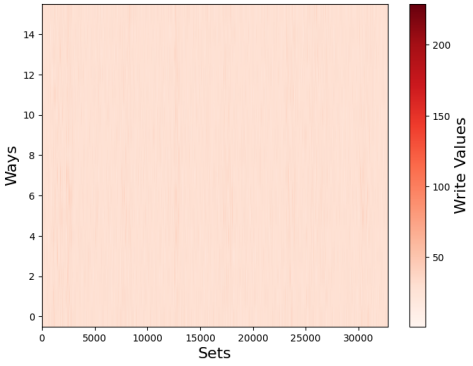

Figure 5.33: SDC:HE-Mix3.

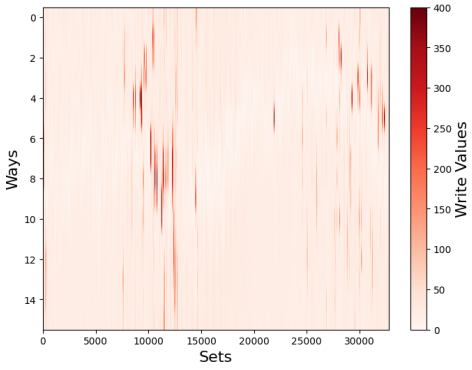

Figure 5.34: LRU-Mix4.

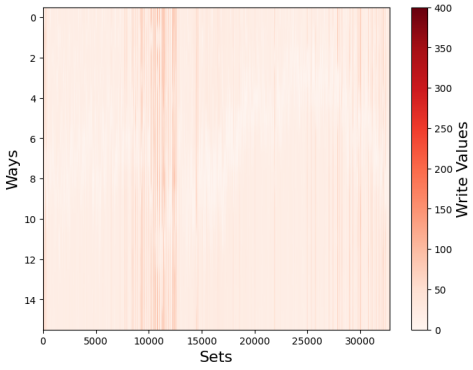

Figure 5.35: i^{2}WAP-Mix4.

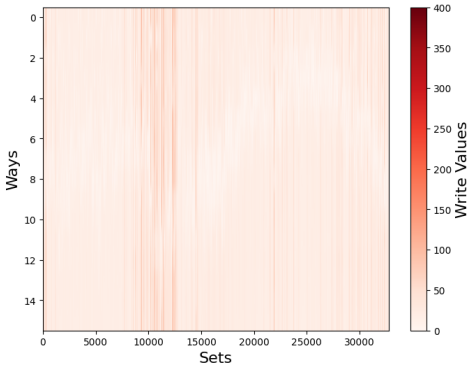

Figure 5.36: SWWR-Mix4.

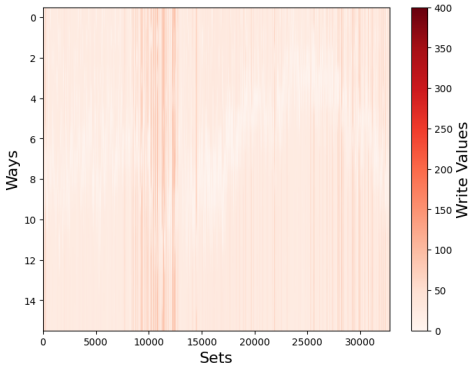

Figure 5.37: DWAWR-Mix4.

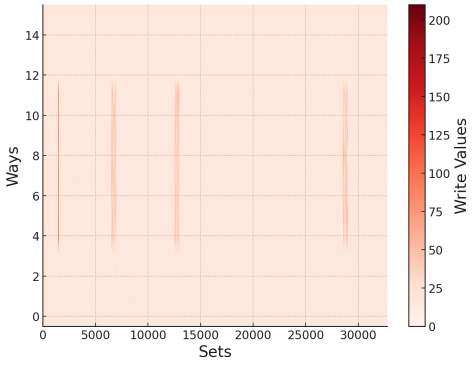

Figure 5.38: PROLONG-Mix4.

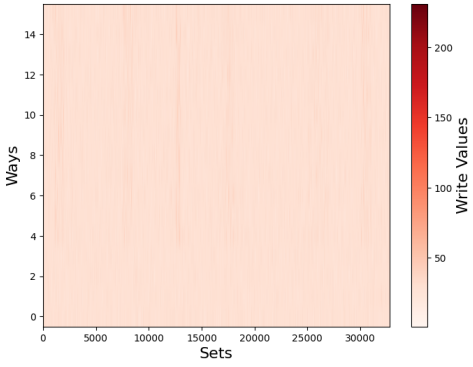

Figure 5.39: LiveWay-Mix4.

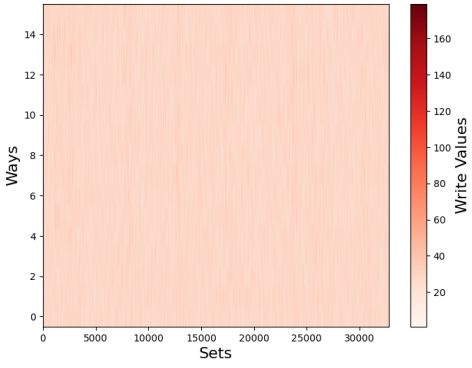

Figure 5.40: SDC:Primal-Mix4.

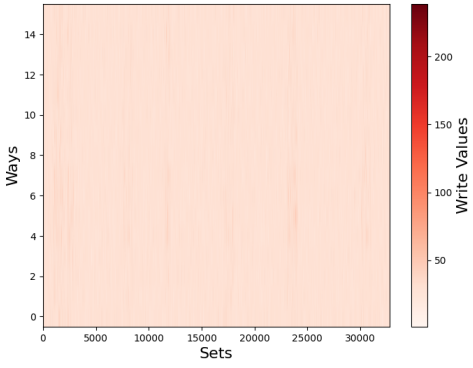

Figure 5.41: SDC:HE-Mix4.

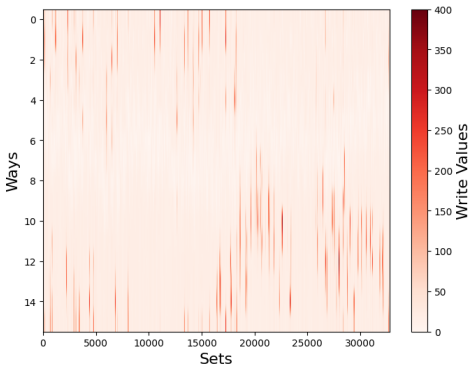

Figure 5.42: LRU-Mix5.

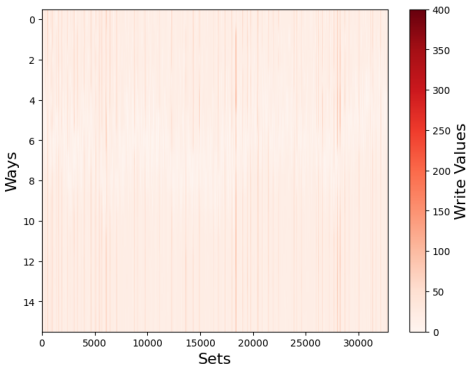

Figure 5.43: i^{2}WAP-Mix5.

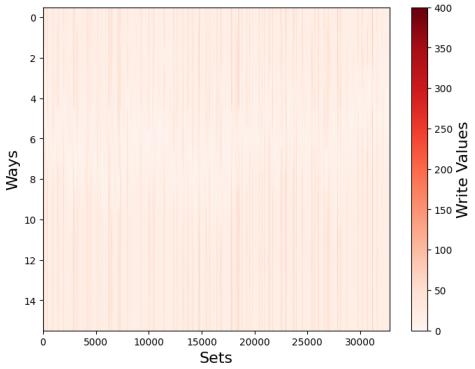

Figure 5.44: SWWR-Mix5.

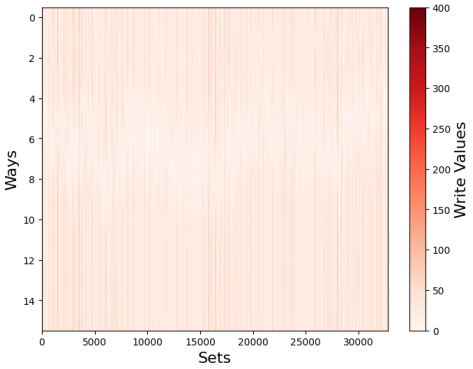

Figure 5.45: DWAWR-Mix5.

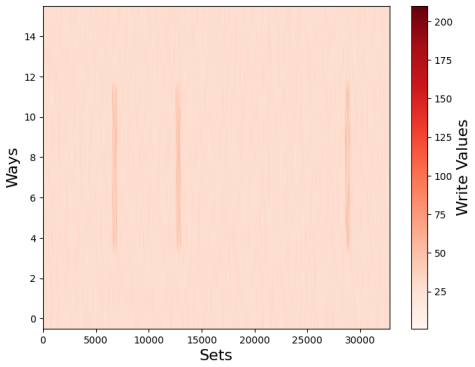

Figure 5.46: PROLONG-Mix5.

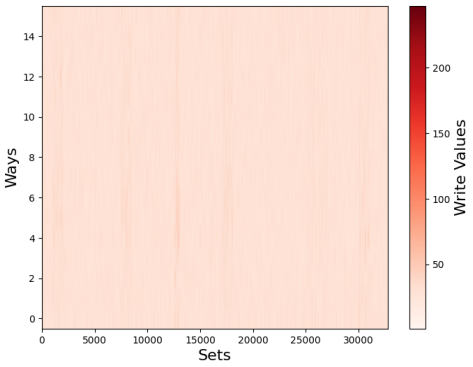

Figure 5.47: LiveWay-Mix5.

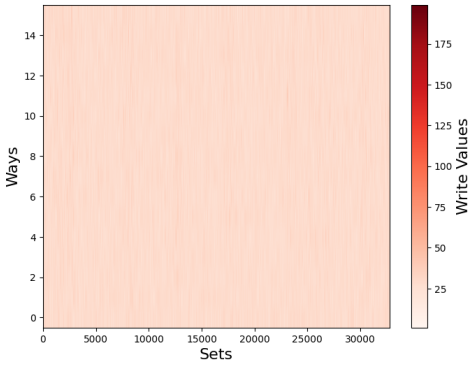

Figure 5.48: SDC:Primal-Mix5.

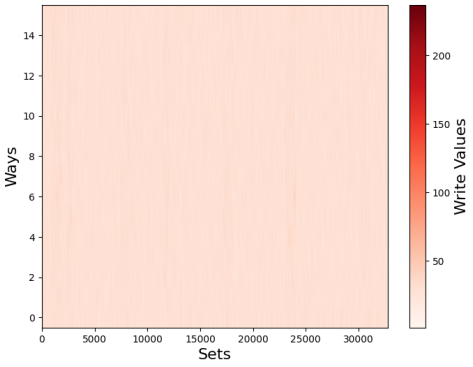

Figure 5.49: SDC:HE-Mix5.

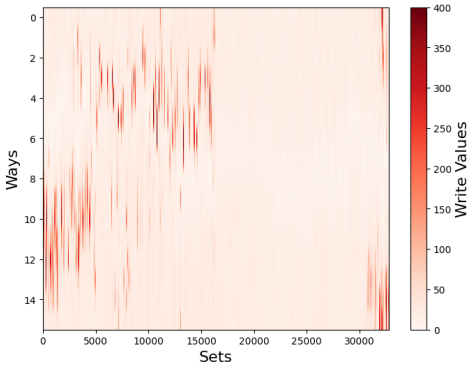

Figure 5.50: LRU-Mix6.

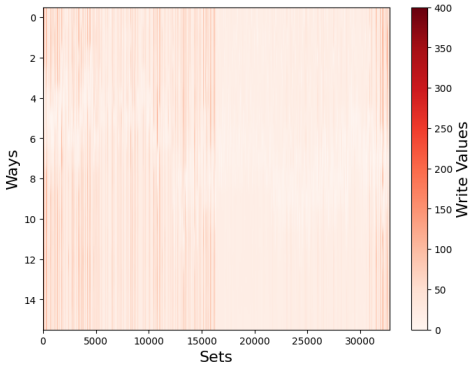

Figure 5.51: i^{2}WAP-Mix6.

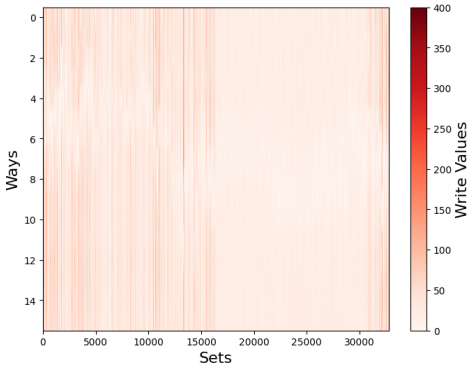

Figure 5.52: SWWR-Mix6.

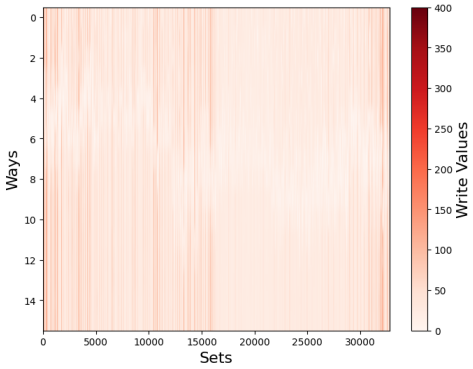

Figure 5.53: DWAWR-Mix6.

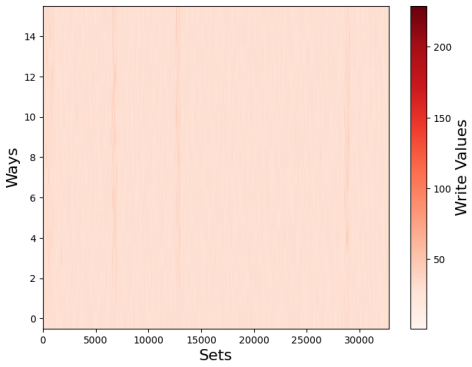

Figure 5.54: PROLONG-Mix6.

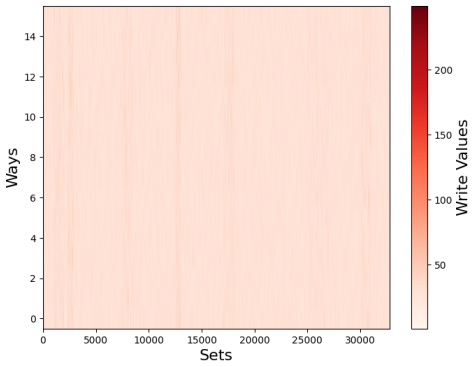

Figure 5.55: LiveWay-Mix6.

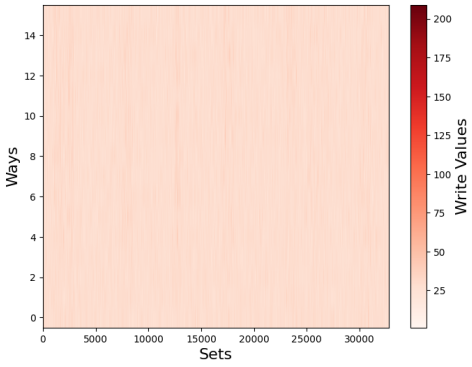

Figure 5.56: SDC:Primal-Mix6.

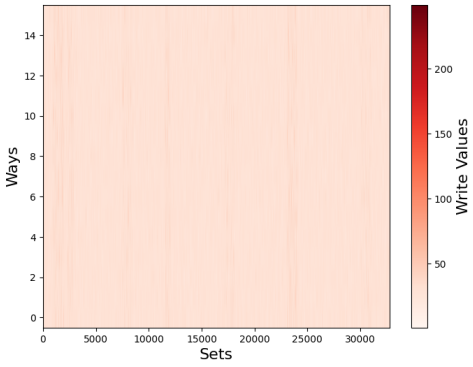

Figure 5.57: SDC:HE-Mix6.

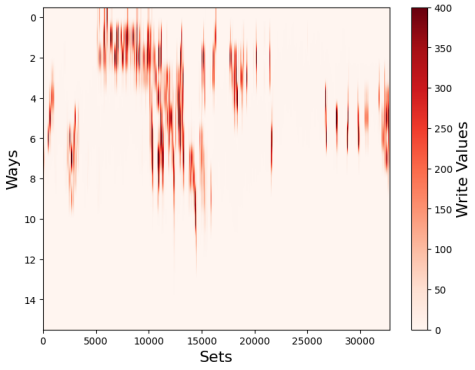

Figure 5.58: LRU-Mix7.

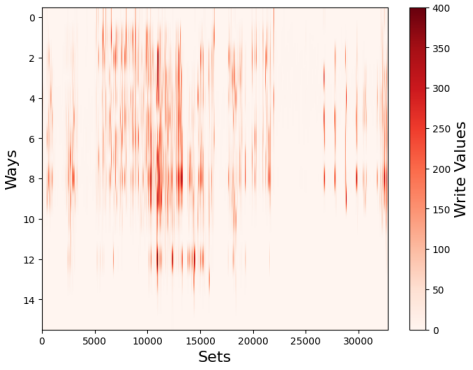

Figure 5.59: i^{2}WAP-Mix7.

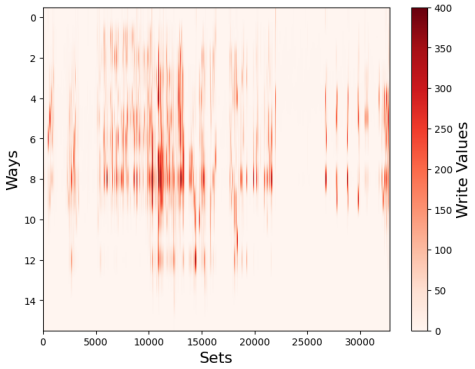

Figure 5.60: SWWR-Mix7.

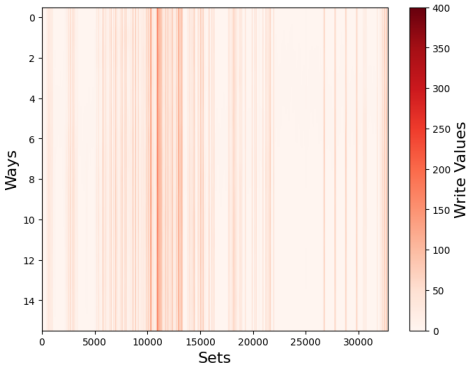

Figure 5.61: DWAWR-Mix7.

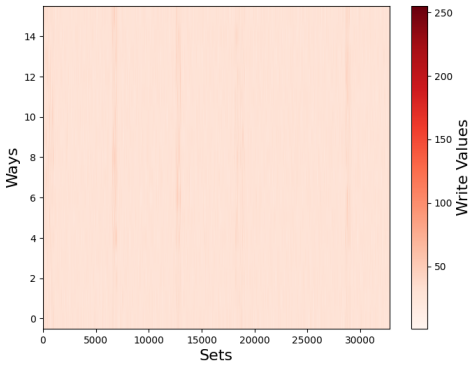

Figure 5.62: PROLONG-Mix7.

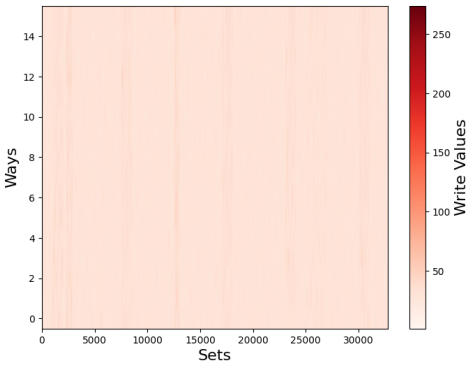

Figure 5.63: LiveWay-Mix7.

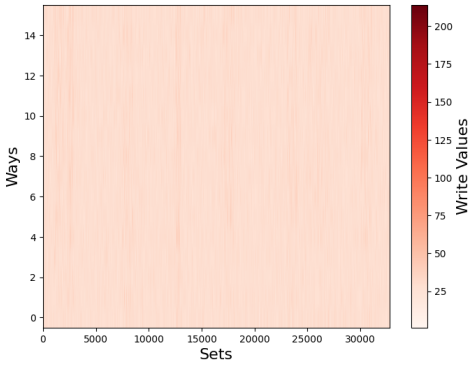

Figure 5.64: SDC:Primal-Mix7.

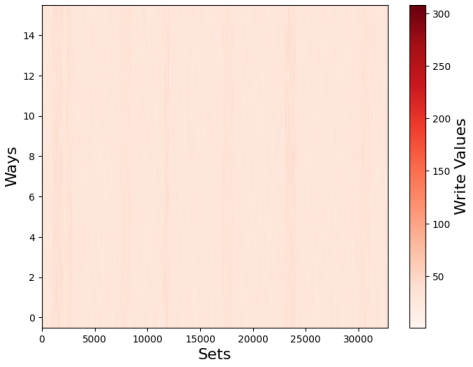

Figure 5.65: SDC:HE-Mix7.

the writes across the LLC to some extent. The Primal and Hardware Efficient that have been proposed is also able to aptly mitigate the write patterns caused by RLA as shown in Figures 5.16 (SDC:Primal-Mix1) and 5.9 (SDC:HE-Mix1).

Figures 5.18, 5.19, 5.20, 5.21, 5.22, 5.23, 5.24, and 5.25 illustrate the impact of RTA on various LLC configurations running on different wear leveling techniques. These figures vividly demonstrate how RTA expands the attack surface by targeting a larger number of locations and dynamically toggling between two sets of attack points. This behavior introduces a heightened intensity of write activity, as the random toggling mechanism effectively undermines traditional counter-based wear-leveling techniques. The randomness and unpredictability of the toggles amplify the challenges faced by these techniques, rendering them less effective in mitigating the wear imbalance caused by such sophisticated attack patterns. The attack patterns are clearly visible with techniques not having any kind of wear leveling. These attack patterns fizzle out with the increasing effectiveness of the wear leveling techniques.

Figures 5.26, 5.27, 5.28, 5.29, 5.30, 5.31, 5.32, and 5.33 illustrate the impact of RLA and RTA when executed concurrently on two cores. The results highlight how these attack patterns diversify across the LLC, significantly amplifying their impact on endurance. The figures further demonstrate how the distribution of attack patterns influences wear across the cache, increasing the potential for early degradation. While counter-based wear-leveling techniques show some effectiveness in mitigating the impacts of RLA and RTA, their ability to fully counteract these attacks remains limited, leaving the system vulnerable to endurance reduction under sustained assault.

To design an attack that is difficult to detect, we propose RnLA and RnTA, leveraging their randomized and dynamic behaviors. Figures 5.34 through 5.41 depict the behavior of the RnLA attack when it is introduced in a single core, while Figures 5.42 through 5.49 illustrate the behavior of the RnTA attack under baseline conditions and with various wear-leveling techniques. For RnLA, the randomization of attack locations results in a significantly uneven distribution of writes, with a notable increase in write intensity. Traditional counter-based wear-leveling techniques are unable to effectively manage this unpredictability. RnTA combines randomization with toggling, creating a highly dynamic and volatile attack pattern that overwhelms these techniques, effectively rendering their countermeasures ineffective in maintaining write balance across the LLC.

When RnLA and RnTA are deployed concurrently in two different cores, the resulting write distribution is illustrated in the heatmaps of Figures 5.50 through 5.57. These figures highlight how the interaction between these attacks leads to a chaotic and uneven write pattern across the LLC. It shows how RnLA and RnTA perform individually and as well as when attacking in tandem through different cores. Furthermore, when all four attacking applications—RLA, RTA, RnLA, and RnTA are simultaneously deployed in four separate cores, as in Mix7, the concentrated and aggressive nature of the attack is vividly captured in Figures 5.58 through 5.65. These heat-maps underscore the significant stress imposed on the STT-RAM LLC, revealing how it behaves under the combined

pressure of an all-out attack. Our analysis also reveals that, apart from a slightly higher number of write operations in AI workloads, they do not introduce any notable ingenuity in multi-core attack scenarios. Instead, they function merely as write-intensive benchmarks without exhibiting distinct write patterns that significantly impact the endurance of an STT-RAM cell. Consequently, we classify them as innocent AI applications, as they do not behave like targeted endurance attacks.

5.5.1.1 Write Count

To further understand the impact of RLA, RTA, RnLA, and RnTA on STT-RAM LLC, Table 5.4 provides the maximum write count recorded at a particular LLC location for different mixes. It can be seen that under baseline Mix0, the maximum write count is 182. This value is reduced to 64, 63, 63, 52, 56, 50, and 54 by the wear-leveling techniques i^{2}WAP, SWWR, DWAWR, PROLONG, LiveWay, SDC:Primal Approach, and SDC:Hadware Efficient Approach respectively, showing their effectiveness. However, with Mix1, the maximum write count significantly increases to 848, 340, 378, 268, 195, 216, 177, and 200 for LRU, i^{2}WAP, SWWR, DWAWR, PROLONG, LiveWay, SDC:Primal Approach, and SDC:Hardware Efficient Approach respectively. This demonstrates the impact of RLA on STT-RAM LLC's lifetime. Mix2 shows the impact of RTA the max write counts. A simple RTA attack leads to a maximum write count of 612 in baseline, which is further reduced respectively by wear eveling techniques. Mix3 shows how the maximum write count is affected when RLA and RTA work in tandem. Mix3 shows a maximum write count of 788 in baseline. The effects of RnLA and RnTA are depicted in Mix4 and Mix5 respectively. In Mix6 when RnLA and RnTA work in tandem with each other, i.e. attacking simultaneously from two different cores then the maximum write count is shown as 625 in baseline. Mix7 shows if a system is completely under attack from different configurations of the endurance attacks, i.e. all four cores are under attack of RLA, RTA, RnLA, and RnTA. It shows that the maximum write count is shown as 864 in baseline, which is further reduced to 694, 590, 344, 256, 27, 215, and 309 for i^{2}WAP, SWWR, DWAWR, PROLONG, LiveWay, SDC:Primal Approach, and SDC:Hardware Efficient Approach respectively.

Table 5.4: The maximum write count of a LLC block.

Mix Variation	LRU	i2wap	SWWR	DWAWR	PROLONG	LiveWay	SDC:Primal	SDC:HE
Mix0	182	64	63	63	52	56	50	54
Mix1(RLA)	848	340	378	268	195	216	177	200
Mix2(RTA)	612	312	374	225	184	205	160	207
Mix3(RLA+RTA)	788	418	569	278	220	240	185	230
Mix4(RnLA)	573	371	291	274	210	232	180	240
Mix5(RnTA)	539	290	265	239	216	238	200	238
Mix6(RnLA+RnTA)	625	304	304	256	230	250	210	251
Mix7(RLA+RTA+RnLA+RnTA)	864	694	590	344	256	275	215	309

5.5.2 Effects on Lifetime and Performance

RTA and RLA contribute to increased write-backs at specific locations in the STT-RAM LLC, whereas RnLA and RnTA target random blocks, leading to a gradual rise in the maximum write count within an LLC block. This, in turn, accelerates wear, significantly reducing the lifetime of the LLC. Lifetime is defined here as the inverse of the maximum write count. Figure 5.66 illustrates the impact of RLA, RTA, RnLA, and RnTA attacks on the LLC lifetime for Baseline (LRU), and other wear leveling techniques like i^{2}WAP, SWWR, DWAWR, PROLONG(10% with 512KB Buffer), LiveWay(128KB Buffer with WindowSize 2), SDC:Primal Approach, and SDC:Hardware Efficent Approach configurations, compared to their respective lifetimes under Mix0 conditions.

From Figure 5.66, it is evident that under the baseline configuration, RLA, RTA, RnLA, and RnTA result in relative lifetime degradations of approximately 78.53%, 70.26%, 68.23%, and 66.23%, respectively. These significant reductions highlight the severe impact of targeted and randomized attacks on the LLC's durability, particularly when wear-leveling techniques fail to mitigate the concentrated or dispersed write patterns introduced by these adversarial workloads.

Existing wear leveling techniques are also severely effected by the RTA attack, with lifetime degradation of 79.48%, 83.15%, 72%, 71.7%, 72.6%, 68.75%, and 73% for i^{2}WAP, SWWR, DWAWR, PROLONG, LiveWay, SDC:Primal Approach, and SDC:Hardware Efficient Approach respectively. The impact of RLA is slightly more than RTA because of its concentrated writes to a particular location(s). With time the effect of RLA will be whittled down by wear leveling techniques as compared to RTA and even RTA will not be highly effective therefore only RnLA and RnTA will be the viable attack options. The increased write count also leads to a drop in the IPC of the system due to the long write latency of STT-RAM [118] leading to congestion at the LLC [13]. Figure 5.67 shows that all the configurations face significant performance degradation due to Mix1-Mix7. Please note that the performance is compared w.r.t. the same configuration when run with Mix0.

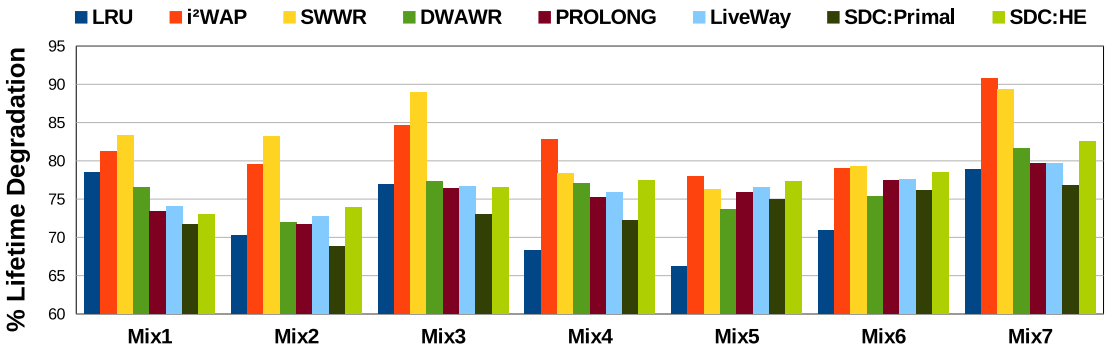

Figure 5.66: Degradation in Lifetime.

IPC degradation ranging from 6% to 45% can be observed among the various configurations needing to deal with RLA and RTA. This shows the effectiveness of both the variations of the attacks in degrading the lifetime and performance of the system.

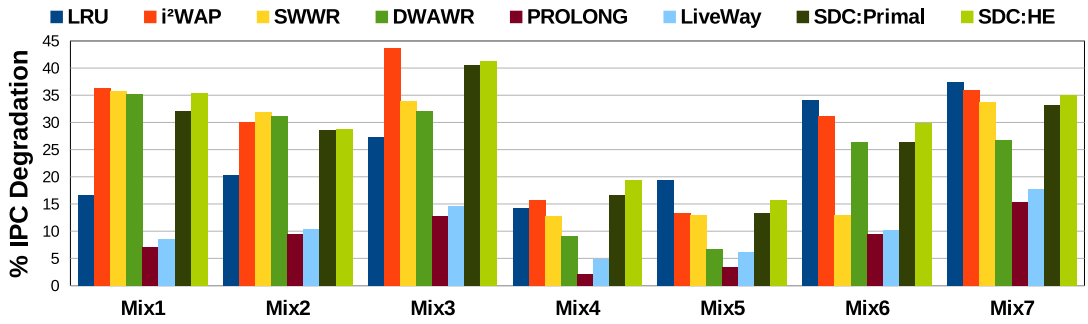

Figure 5.67: Degradation in IPC.

5.6 Conclusion and Future Work

Through this work we have shown the vulnerability of a shared STT-RAM LLC because of targeted endurance attacks. Although wear leveling techniques are still impactful in distributing the writes even with endurance attacks, the lifetime of LLC drops by 84.68%, 88.92%, and 77.33% for i^{2}WAP, SWWR and DWAWR, respectively. Since limited endurance is a significant challenge for STT-RAM LLCs; RLA, RTA, RnLA and RnTA type endurance attacks pose a major concern. The current wear-leveling techniques are not effective in preventing these attacks, highlighting the need for more sophisticated approaches. Rather than relying solely on counter-based or invalidation-based methods, an intelligent write-bypassing technique can be integrated with them. Exploring variations of these attacks and developing advanced defense techniques are part of our future research.

Chapter 6

Conclusion and Future Scope

This chapter presents a comprehensive conclusion of the research detailed in various chapters of this thesis. Section 6.1 offers a summary of the key findings and draws an overall conclusion, followed by the scope of future work in Section 6.2.

6.1 Major Research Contributions

Figure 6.1 gives us a birds eye view of the thesis contributions and the effects they have on STT-RAM LLCs.

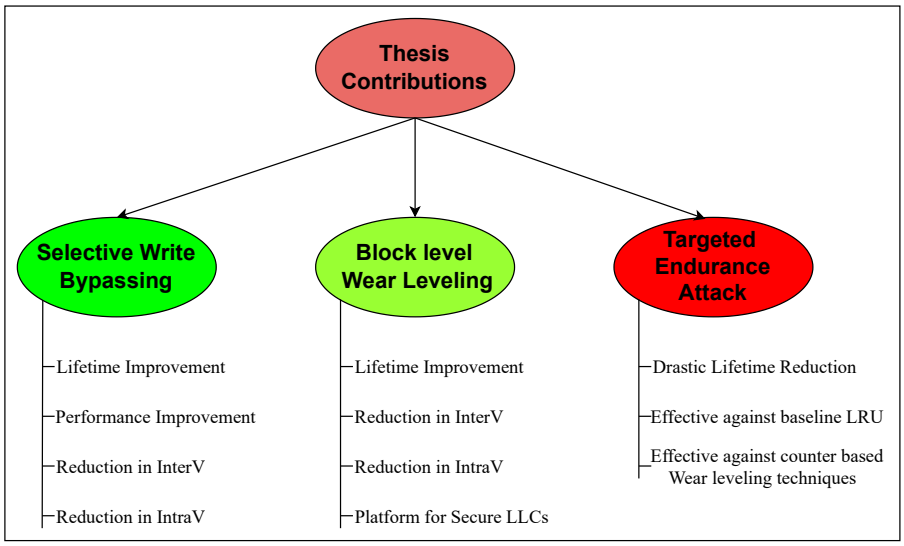

Figure 6.1: Graphical Overview of Thesis

The following are major research contributions discussed in this thesis:

– The first work exposes the write imbalance across the STT-RAM LLC that results in heightened InterV and IntraV which is detrimental to the lifetime of the STT-RAM LLC. In order to mitigate this problem of limited write endurance, without causing too much performance overhead, we proposed two different techniques: PROLONG: Priority based Write Bypassing Technique for Longer Lifetime in STT-RAM based LLC that is aimed at reducing the InterV and increasing the Lifetime and LiveWay: Dynamic Write Bypassing for Lifetime Enhancement in STT-RAM LLC that aims at reducing the IntraV and increasing the lifetime without putting undue stress on the performance of the system.

- The second work highlights the write imbalance present among all the blocks of an STT-RAM LLC which leads to lifetime degradation and the need for a block-level wear leveling technique. It highlights that most of the wear-leveling techniques proposed were either on the set-level granularity or way-level granularity. Therefore, we propose a a novel decoupled tag and data array architecture with an Optimal wear-leveling technique that distributes the writes across all the blocks in the STT-RAM based LLC. Write hot blocks are identified, and then the redirection takes place by exchanging the tag and data array blocks. A hardware efficient approach is also proposed that limits the number of counters by assigning a single counter to a bucket and each bucket consists of a group of blocks.

- The third work highlights the vulnerability of the STT-RAM based LLC to targeted write attacks. To this end, we propose four different kinds of targeted endurance attacks (TENDRA): RLA: It repeatedly writes a single location in the STT-RAM LLC; RTA: It repeatedly toggles writes in between two different locations in the LLC; RnLA: It randomly targets different location to write to random amount of times; and RnTA: It randomly selects two different locations to target writes to random amount of times.

6.2 Future Scope

Irrespective of the challenges in implementing STT-RAM as shared LLCs in a generic computing system, the day is not far off when it can be implemented successfully. If implemented it will prove to be a big advantage, especially with its high density and low power consumption capabilities. Some future research directions are as follows:

- One of the major challenges in implementing STT-RAM LLCs is the different read and write latencies. Servicing writes from the read-write queue causes congestion and therefore leads to drastic performance degradation as compared to a generic SRAM implementation. Furthermore, uneven writes can create heightened WV in the STT-RAM LLCs which will lead to lifetime degradation. Therefore, there is a need to propose techniques that aim to reduce both the lifetime degradation and performance degradation through effective wear leveling and intelligent write bypassing.

- In a multi-core system having a single endurance attack can compromise the integrity and lifetime of the whole STT-RAM LLC as targeted endurance attacks make the system vulnerable. The lifetime of the LLC is drastically effected. In order to mitigate these attacks we need to propose proper attack identification and mitigation techniques that are able to dead with these problems of attacks from a single compromised core.

6.2.1 Practical Lifetime Improvement for Industry-wide Adaptation

To replace SRAM as the dominant CMOS design for last-level caches (LLCs), STT-RAM must match or surpass nearly all performance metrics of a standard SRAM-based LLC. Although, STT-RAM offers numerous advantages as a potential replacement, its drawbacks must not compromise the overall system functionality. In particular, endurance remains a critical challenge, which requires strategies to maximize the longevity of STT-RAM-based LLCs. Our state-of-the-art STT-RAM LLC designs have demonstrated up to a 34× improvement in lifetime.

For STT-RAM to be a viable alternative, it must not only achieve endurance levels comparable to those of SRAM but also mitigate performance degradation due to increased write latency, which reduces IPC. Kargaokar et al. [13] highlight this issue, showing that even with optimization techniques, an 8MB STT-RAM cache struggles to match the performance of a 4MB SRAM cache. This underscores the need for significant advances in both lifetime improvement and write latency reduction before STT-RAM can be seriously considered as a replacement.

Our thesis contributes to this effort by enhancing the endurance of STT-RAM-based LLCs while also exploring how endurance vulnerabilities can be exploited by attackers. This work represents a step forward in making STT-RAM-based LLCs a feasible alternative to SRAM-based designs.

References

[1] Naveen Muralimanohar, Rajeev Balasubramonian, and Norm Jouppi. Optimizing NUCA Organizations and Wiring Alternatives for Large Caches with CACTI 6.0. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 40, pages 3–14, 2007. ISBN 0-7695-3047-8.

[2] Rajeev Balasubramonian, Norman P Jouppi, and Naveen Muralimanohar. Multi-core cache hierarchies. *Synthesis Lectures on Computer Architecture*, 6(3): 1–153, 2011.

[3] Peter Kogge, S. Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Jon Hiller, Stephen Keckler, Dean Klein, and Robert Lucas. Exascale computing study: Technology challenges in achieving exascale systems. *Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Techinal Representative*, 15, 01 2008.

[4] Chris H. Kim, Jae-Joon Kim, Saibal Mukhopadhyay, and Kaushik 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*, ISLPED '03, page 6–9, New York, NY, USA, 2003. Association for Computing Machinery. ISBN 158113682X. doi: 10.1145/871506.871511. URL https://doi.org/10.1145/871506.871511.

[5] S. S. Iyer, J. E. Barth, P. C. Parries, J. P. Norum, J. P. Rice, L. R. Logan, and D. Hoyniak. Embedded dram: Technology platform for the blue gene/l chip. *IBM Journal of Research and Development*, 49(2.3):333–350, 2005. doi: 10.1147/rd.492.0333.

[6] Sadegh Yazdanshenas, Marzieh Ranjbar Pirbasti, Mahdi Fazeli, and Ahmad Patooghy. Coding last level stt-ram cache for high endurance and low power. *IEEE Computer Architecture Letters*, 13(2):73–76, 2014. doi: 10.1109/L-CA.2013.8.

[7] Young-Bae Kim, Seung Ryul Lee, Dongsoo Lee, Chang Bum Lee, Man Chang, Ji Hyun Hur, Myoung-Jae Lee, Gyeong Su Park, Chang Jung Kim, U-In Chung, In kyeong Yoo, and Kinam Kim. Bi-layered rram with unlimited endurance and extremely uniform switching. *2011 Symposium on VLSI Technology - Digest of Technical Papers*, pages 52–53, 2011. URL https://api.semanticscholar.org/CorpusID:44515793.

[8] Moinuddin Qureshi, Sudhanva Gurumurthi, and Bipin Rajendran. *Phase Change Memory: From Devices to Systems*, volume 6. Morgan & Claypool Publishers, 11 2011. doi: 10.2200/S00381ED1V01Y201109CAC018.

[9] Rangharajan Venkatesan, Mrigank Sharad, Kaushik Roy, and Anand Raghunathan. Dwm-tapestri - an energy efficient all-spin cache using domain wall shift based writes. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '13, page 1825–1830, San Jose, CA, USA, 2013. EDA Consortium. ISBN 9781450321532.

[10] Sparsh Mittal, Jeffrey S. Vetter, and Dong Li. A survey of architectural approaches for managing embedded dram and non-volatile on-chip caches. *IEEE Transactions on Parallel and Distributed Systems*, 26(6):1524–1537, 2015. doi: 10.1109/TPDS.2014.2324563.

[11] Jimmy J. Kan, Chando Park, Chi Ching, Jaesoo Ahn, Yuan Xie, Mahendra Pakala, and Seung H. Kang. A study on practically unlimited endurance of stt-mram. *IEEE Transactions on Electron Devices*, 64(9):3639–3646, 2017. doi: 10.1109/TED.2017.2731959.

[12] Jue Wang, Xiangyu Dong, Yuan Xie, and Norman 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)*, pages 234–245, 2013. doi: 10.1109/HPCA.2013.6522322.

[13] Kunal Korgaonkar, Ishwar Bhati, Huichu Liu, Jayesh Gaur, Sasikanth Manipatruni, Sreenivas Subramoney, Tanay Karnik, Steven Swanson, Ian Young, and Hong Wang. Density tradeoffs of non-volatile memory as a replacement for sram based last level cache. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 315–327, 2018. doi: 10.1109/ISCA.2018.00035.

[14] Sukarn Agarwal and Hemangee K. Kapoor. Improving the lifetime of non-volatile cache by write restriction. *IEEE Transactions on Computers*, 68(9):1297–1312, 2019. doi: 10.1109/TC.2019.2892424.

[15] Mohammad Reza Jokar, Mohammad Arjomand, and Hamid Sarbazi-Azad. Sequoia: A high-endurance nvm-based cache architecture. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(3):954–967, 2016. doi: 10.1109/TVLSI.2015.2420954.

[16] Sparsh Mittal and Jeffrey S. Vetter. EqualChance: Addressing intra-set write variation to increase lifetime of non-volatile caches. In *2nd Workshop on Interactions of NVM/Flash with Operating Systems and Workloads (INFLOW 14)*, Broomfield, CO, October 2014. USENIX Association. URL https://www.usenix.org/conference/inflow14/workshop-program/presentation/mittal.

[17] Sparsh Mittal, Jeffrey S. Vetter, and Dong 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*, GLSVLSI '14, page 139–144, New York,

NY, USA, 2014. Association for Computing Machinery. ISBN 9781450328166. doi: 10.1145/2591513.2591525. URL https://doi.org/10.1145/2591513.2591525.

[18] Sparsh Mittal, Jeffrey S. Vetter, and Dong Li. Lastingnvcache: A technique for improving the lifetime of non-volatile caches. In *2014 IEEE Computer Society Annual Symposium on VLSI*, pages 534–540, 2014. doi: 10.1109/ISVLSI.2014.69.

[19] Chao Zhang, Guangyu Sun, Peng Li, Tao Wang, Dimin Niu, and Yiran Chen. Sbac: A statistics based cache bypassing method for asymmetric-access caches. In *2014 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 345–350, 2014. doi: 10.1145/2627369.2627611.

[20] Junwhan Ahn, Sungjoo Yoo, and Kiyoung Choi. Dasca: Dead write prediction assisted stt-ram cache architecture. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, pages 25–36, 2014. doi: 10.1109/HPCA.2014.6835944.

[21] Min Kyu Kim, Ju Hee Choi, Jong Wook Kwak, Seong Tae Jhang, and Chu Shik Jhon. Bypassing method for stt-ram based inclusive last-level cache. In *Proceedings of the 2015 Conference on Research in Adaptive and Convergent Systems*, RACS '15, page 424–429, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450337380. doi: 10.1145/2811411.2811512. URL https://doi.org/10.1145/2811411.2811512.

[22] Aritra Bagchi, Dharamjeet, Ohm Rishabh, Manan Suri, and Preeti Ranjan Panda. Poem: Performance optimization and endurance management for non-volatile caches. *ACM Trans. Des. Autom. Electron. Syst.*, 29(5), September 2024. ISSN 1084-4309. doi: 10.1145/3653452. URL https://doi.org/10.1145/3653452.

[23] Xiaoxia Wu, Jian Li, Lixin Zhang, Evan Speight, Ram Rajamony, and Yuan Xie. Hybrid cache architecture with disparate memory technologies. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, page 34–45, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585260. doi: 10.1145/1555754.1555761. URL https://doi.org/10.1145/1555754.1555761.

[24] Adwait Jog, Asit K. Mishra, Cong Xu, Yuan Xie, Vijaykrishnan Narayanan, Ravishankar Iyer, and Chita R. Das. Cache revive: Architecting volatile stt-ram caches for enhanced performance in cmps. In *DAC Design Automation Conference 2012*, pages 243–252, 2012. doi: 10.1145/2228360.2228406.

[25] Baixing Quan, Tiefei Zhang, Tianzhou Chen, and Jianzhong Wu. Prediction table based management policy for stt-ram and sram hybrid cache. In *2012 7th International Conference on Computing and Convergence Technology (ICCCT)*, pages 1092–1097, 2012.

[26] Sparsh Mittal and Jeffrey S. Vetter. Ayush: A technique for extending lifetime of sram-nvm hybrid caches. *IEEE Computer Architecture Letters*, 14(2):115–118, 2015. doi: 10.1109/LCA.2014.2355193.

[27] Ning Bao, Yun-Peng Chai, Xiao Qin, and Chuan-Wen Wang. Macrotrend: A write-efficient cache algorithm for nvm-based read cache. *Journal of Computer Science and Technology*, 37(1):207–230, 2022. doi: 10.1007/s11390-021-0178-6. URL https://www.sciopen.com/article/10.1007/s11390-021-0178-6.

[28] Dhruv Gajaria, Kevin Antony Gomez, and Tosiron Adegbija. A study of stt-ram-based in-memory computing across the memory hierarchy. In *2022 IEEE 40th International Conference on Computer Design (ICCD)*, pages 685–692, 2022. doi: 10.1109/ICCD56317.2022.00105.

[29] Kyle Kuan and Tosiron Adegbija. Halls: An energy-efficient highly adaptable last level stt-ram cache for multicore systems. *IEEE Trans. Comput.*, 68(11):1623–1634, nov 2019. ISSN 0018-9340. doi: 10.1109/TC.2019.2918153. URL https://doi.org/10.1109/TC.2019.2918153.

[30] Puneet Saraf and Madhu Mutyam. Endurance enhancement of write-optimized stt-ram caches. In *Proceedings of the International Symposium on Memory Systems*, MEMSYS '19, page 101–113, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450372060. doi: 10.1145/3357526.3357538. URL https://doi.org/10.1145/3357526.3357538.

[31] Saeed Kargar and Faisal Nawab. Extending the lifetime of nvm: challenges and opportunities. *Proc. VLDB Endow.*, 14(12):3194–3197, July 2021. ISSN 2150-8097. doi: 10.14778/3476311.3476406. URL https://doi.org/10.14778/3476311.3476406.

[32] Zhaoxia Deng, Lunkai Zhang, Nikita Mishra, Henry Hoffmann, and Frederic T. Chong. Memory cocktail therapy: A general learning-based framework to optimize dynamic tradeoffs in nvms. In *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 232–244, 2017.

[33] Saeed Kargar and Faisal Nawab. Challenges and future directions for energy, latency, and lifetime improvements in nvms. *Distrib. Parallel Databases*, 41(3): 163–189, September 2022. ISSN 0926-8782. doi: 10.1007/s10619-022-07421-x. URL https://doi.org/10.1007/s10619-022-07421-x.

[34] Sparsh Mittal. Using cache-coloring to mitigate inter-set write variation in non-volatile caches, 2013.

[35] Sparsh Mittal and Jeffrey S. Vetter. Equalwrites: Reducing intra-set write variations for enhancing lifetime of non-volatile caches. *IEEE Transactions on*

Very Large Scale Integration (VLSI) Systems, 24(1):103–114, 2016. doi: 10.1109/TVLSI.2015.2389113.

[36] Sukarn Agarwal and Hemangee K. Kapoor. Targeting inter set write variation to improve the lifetime of non-volatile cache using fellow sets. In *2017 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 1–6, 2017. doi: 10.1109/VLSI-SoC.2017.8203453.

[37] Sukarn Agarwal and Hemangee K. Kapoor. Lifetime enhancement of non-volatile caches by exploiting dynamic associativity management techniques. In Michail Maniatakos, Ibrahim (Abe) M. Elfadel, Matteo Sonza Reorda, H. Fatih Ugurdag, José Monteiro, and Ricardo Reis, editors, *VLSI-SoC: Opportunities and Challenges Beyond the Internet of Things*, pages 46–71, Cham, 2019. Springer International Publishing. ISBN 978-3-030-15663-3.

[38] Morteza Soltani, Mohammad Ebrahimi, and Zainalabedin Navabi. Prolonging lifetime of non-volatile last level caches with cluster mapping. In *2016 International Great Lakes Symposium on VLSI (GLSVLSI)*, pages 329–334, 2016. doi: 10.1145/2902961.2902980.

[39] S. Sivakumar, T.M. Abdul Khader, and John Jose. Improving lifetime of non-volatile memory caches by logical partitioning. In *Proceedings of the 2021 Great Lakes Symposium on VLSI*, GLSVLSI '21, page 123–128, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383936. doi: 10.1145/3453688.3461488. URL https://doi.org/10.1145/3453688.3461488.

[40] S. Sivakumar and John Jose. Self adaptive logical split cache techniques for delayed aging of nvm llc. *ACM Trans. Des. Autom. Electron. Syst.*, 28(6), October 2023. ISSN 1084-4309. doi: 10.1145/3616871. URL https://doi.org/10.1145/3616871.

[41] S. Sivakumar, Mani Mannampalli, and John Jose. Enhancing lifetime of non-volatile memory caches by write-aware techniques. In Chandan Giri, Takahiro Iizuka, Hafizur Rahaman, and Bhargab B. Bhattacharya, editors, *Emerging Electronic Devices, Circuits and Systems*, pages 109–123, Singapore, 2023. Springer Nature Singapore. ISBN 978-981-99-0055-8.

[42] Sparsh Mittal. A survey of cache bypassing techniques. *Journal of Low Power Electronics and Applications*, 6(2), 2016. ISSN 2079-9268. doi: 10.3390/jlpea6020005. URL https://www.mdpi.com/2079-9268/6/2/5.

[43] Haakon Dybdahl and Per Stenström. Enhancing last-level cache performance by block bypassing and early miss determination. In *Proceedings of the 11th Asia-Pacific Conference on Advances in Computer Systems Architecture*, ACSAC'06, page 52–66, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 3540400567. doi: 10.1007/11859802_6. URL https://doi.org/10.1007/11859802_6.

[44] Jue Wang, Xiangyu Dong, and Yuan Xie. Oap: An obstruction-aware cache management policy for stt-ram last-level caches. *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 847–852, 2013. URL https://api.semanticscholar.org/CorpusID:9961439.

[45] Ing-Chao Lin and Jeng-Nian Chiou. High-endurance hybrid cache design in cmp architecture with cache partitioning and access-aware policies. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(10):2149–2161, 2015. doi: 10.1109/TVLSI.2014.2361150.

[46] Zhe Wang, Daniel A. Jiménez, Cong Xu, Guangyu Sun, and Yuan Xie. Adaptive placement and migration policy for an stt-ram-based hybrid cache. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, pages 13–24, 2014. doi: 10.1109/HPCA.2014.6835933.

[47] Swapnil Bhosale and Sudeep Pasricha. Slam: High performance and energy efficient hybrid last level cache architecture for multicore embedded systems. In *2019 IEEE International Conference on Embedded Software and Systems (ICESS)*, pages 1–7, 2019. doi: 10.1109/ICESS.2019.8782441.

[48] Sparsh Mittal and Jeffrey Vetter. A technique for improving lifetime of non-volatile caches using write-minimization. *Journal of Low Power Electronics and Applications*, 6(1), 2016. ISSN 2079-9268. doi: 10.3390/jlpea6010001. URL https://www.mdpi.com/2079-9268/6/1/1.

[49] Yongsoo Joo, Dimin Niu, Xiangyu Dong, Guangyu Sun, Naehyuck Chang, and Yuan Xie. Energy- and endurance-aware design of phase change memory caches. In *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, pages 136–141, 2010. doi: 10.1109/DATE.2010.5457221.

[50] Sina Asadi, Amir Mahdi Hosseini Monazzah, Hamed Farbeh, and Seyed Ghassem Miremadi. Wipe: Wearout informed pattern elimination to improve the endurance of nvm-based caches. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 188–193, 2017. doi: 10.1109/ASPDAC.2017.7858318.

[51] S. Sivakumar, John Jose, and Vijaykrishnan Narayanan. Enhancing lifetime and performance of mlc nvm caches using embedded trace buffers. *ACM Trans. Des. Autom. Electron. Syst.*, 29(3), May 2024. ISSN 1084-4309. doi: 10.1145/3659102. URL https://doi.org/10.1145/3659102.

[52] Bhukya Krishna Priya. Enhancing the lifetime of stt-ram using compression based wear leveling technique. *Microelectronics Reliability*, 143:114939, 2023. ISSN 0026-2714. doi: https://doi.org/10.1016/j.microrel.2023.114939. URL https://www.sciencedirect.com/science/article/pii/S0026271423000392.

[53] Shuai Wang, Guangshan Duan, Yupeng Li, and Qianhao Dong. Word- and partition-level write variation reduction for improving non-volatile cache lifetime. *ACM Trans. Des. Autom. Electron. Syst.*, 23(1), August 2017. ISSN 1084-4309. doi: 10.1145/3084690. URL https://doi.org/10.1145/3084690.

[54] Gorka Irazoqui Apecechea et al. S$A: A Shared Cache Attack That Works across Cores and Defies VM Sandboxing – and Its Application to AES. *IEEE Symp. on Security and Privacy*, pages 591–604, 2015.

[55] Ahmad Moghimi, Gorka Irazoqui Apecechea, and Thomas Eisenbarth. Cachezoom: How sgx amplifies the power of cache attacks. *ArXiv*, abs/1703.06986, 2017. URL https://api.semanticscholar.org/CorpusID:6116879.

[56] Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Cross-vm side channels and their use to extract private keys. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, page 305–316, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450316514. doi: 10.1145/2382196.2382230. URL https://doi.org/10.1145/2382196.2382230.

[57] Jaspinder Kaur and Shirshendu Das. Acpc: Covert channel attack on last level cache using dynamic cache partitioning. In *2023 24th International Symposium on Quality Electronic Design (ISQED)*, pages 1–8, 2023. doi: 10.1109/ISQED57927.2023.10129363.

[58] Clémentine Maurice, Christoph Neumann, Olivier Heen, and Aurélien Francillon. C5: Cross-cores cache covert channel. In Magnus Almgren, Vincenzo Gulisano, and Federico Maggi, editors, *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 46–64, Cham, 2015. Springer International Publishing. ISBN 978-3-319-20550-2.

[59] Fan Yao, Miloš Doroslovački, and Guru Venkataramani. Covert timing channels exploiting cache coherence hardware: Characterization and defense. *Int. J. Parallel Program.*, 47(4):595–620, August 2019. ISSN 0885-7458. doi: 10.1007/s10766-018-0608-4. URL https://doi.org/10.1007/s10766-018-0608-4.

[60] Craig Disselkoen, David Kohlbrenner, Leo Porter, and Dean Tullsen. Prime+abort: a timer-free high-precision l3 cache attack using intel tsx. In *Proceedings of the 26th USENIX Conference on Security Symposium*, SEC'17, page 51–67, USA, 2017. USENIX Association. ISBN 9781931971409.

[61] Yossef Oren, Vasileios P. Kemerlis, Simha Sethumadhavan, and Angelos D. Keromytis. The spy in the sandbox: Practical cache attacks in javascript and their implications. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, page 1406–1418, New York,

NY, USA, 2015. Association for Computing Machinery. ISBN 9781450338325. doi: 10.1145/2810103.2813708. URL https://doi.org/10.1145/2810103.2813708.

[62] Anirban Chakraborty, Nimish Mishra, Sayandeep Saha, Sarani Bhattacharya, and Debdeep Mukhopadhyay. On the amplification of cache occupancy attacks in randomized cache architectures, 2023. URL https://arxiv.org/abs/2310.05172.

[63] Gururaj Saileshwar and Moinuddin Qureshi. MIRAGE: Mitigating Conflict-Based cache attacks with a practical Fully-Associative design. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1379–1396. USENIX Association, August 2021. ISBN 978-1-939133-24-3. URL https://www.usenix.org/conference/usenixsecurity21/presentation/saileshwar.

[64] Anatoly Shusterman, Lachlan Kang, Yarden Haskal, Yosef Meltser, Prateek Mittal, Yossi Oren, and Yuval Yarom. Robust website fingerprinting through the cache occupancy channel. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 639–656, Santa Clara, CA, August 2019. USENIX Association. ISBN 978-1-939133-06-9. URL https://www.usenix.org/conference/usenixsecurity19/presentation/shusterman.

[65] Anatoly Shusterman, Zohar Avraham, Eliezer Croitoru, Yarden Haskal, Lachlan Kang, Dvir Levi, Yosef Meltser, Prateek Mittal, Yossi Oren, and Yuval Yarom. Website fingerprinting through the cache occupancy channel and its real world practicality. *IEEE Transactions on Dependable and Secure Computing*, 18(5): 2042–2060, 2021. doi: 10.1109/TDSC.2020.2988369.

[66] Mohammad Nasim Imtiaz Khan and Swaroop Ghosh. Analysis of row hammer attack on sttram. In *2018 IEEE 36th International Conference on Computer Design (ICCD)*, pages 75–82, 2018. doi: 10.1109/ICCD.2018.00021.

[67] Felix Staudigl, Jan Philipp Thoma, Christian Niesler, Karl Sturm, Rebecca Pelke, Dominik Germek, Jan Moritz Joseph, Tim Güneysu, Lucas Davi, and Rainer Leupers. Nvm-flip: Non-volatile-memory bitflips on the system level. In *Proceedings of the 2024 ACM Workshop on Secure and Trustworthy Cyber-Physical Systems*, SaT-CPS '24, page 11–20, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400705557. doi: 10.1145/3643650.3658606. URL https://doi.org/10.1145/3643650.3658606.

[68] Felix Staudigl, Hazem Al Indari, Daniel Schön, Dominik Sisejkovic, Farhad Merchant, Jan Moritz Joseph, Vikas Rana, Stephan Menzel, and Rainer Leupers. Neurohammer: inducing bit-flips in memristive crossbar memories. In *Proceedings of the 2022 Conference & Exhibition on Design, Automation & Test in Europe*, DATE '22, page 1181–1184, Leuven, BEL, 2022. European Design and Automation Association. ISBN 9783981926361.

[69] Moinuddin K. Qureshi. New attacks and defense for encrypted-address cache. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, pages 360–371, 2019.

[70] Moinuddin K. Qureshi. Ceaser: mitigating conflict-based cache attacks via encrypted-address and remapping. In *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-51, page 775–787. IEEE Press, 2018. ISBN 9781538662403. doi: 10.1109/MICRO.2018.00068. URL https://doi.org/10.1109/MICRO.2018.00068.

[71] Daniel Gruss, Raphael Spreitzer, and Stefan Mangard. Cache template attacks: Automating attacks on inclusive Last-Level caches. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 897–912, Washington, D.C., August 2015. USENIX Association. ISBN 978-1-939133-11-3. URL https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/gruss.

[72] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. Software grand exposure: SGX cache attacks are practical. In *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, Vancouver, BC, August 2017. USENIX Association. URL https://www.usenix.org/conference/woot17/workshop-program/presentation/brasser.

[73] Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, and Stefan Mangard. Malware guard extension: Using sgx to conceal cache attacks. In Michalis Polychronakis and Michael Meier, editors, *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 3–24, Cham, 2017. Springer International Publishing. ISBN 978-3-319-60876-1.

[74] Mehmet Sinan İnci, Berk Gulmezoglu, Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. Cache attacks enable bulk key recovery on the cloud. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems – CHES 2016*, pages 368–388, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg. ISBN 978-3-662-53140-2.

[75] Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. Cache attacks on intel sgx. In *Proceedings of the 10th European Workshop on Systems Security*, EuroSec'17, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349352. doi: 10.1145/3065913.3065915. URL https://doi.org/10.1145/3065913.3065915.

[76] David Gullasch, Endre Bangerter, and Stephan Krenn. Cache games – bringing access-based cache attacks on aes to practice. In *2011 IEEE Symposium on Security and Privacy*, pages 490–505, 2011. doi: 10.1109/SP.2011.22.

[77] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. Last-Level Cache Side-Channel Attacks are Practical. In *Symp. on Security and Privacy*, pages 605–622, 2015.

[78] Onur Acıiçmez, Billy Bob Brumley, and Philipp Grabher. New results on instruction cache attacks. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010*, pages 110–124, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-15031-9.

[79] Daimeng Wang, Zhiyun Qian, Nael Abu-Ghazaleh, and Srikanth V. Krishnamurthy. Papp: Prefetcher-aware prime and probe side-channel attack. In *Proceedings of the 56th Annual Design Automation Conference 2019*, DAC '19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367257. doi: 10.1145/3316781.3317877. URL https://doi.org/10.1145/3316781.3317877.

[80] Daniel Gruss, Clémentine Maurice, Klaus Wagner, and Stefan Mangard. Flush+Flush: A Fast and Stealthy Cache Attack. In *Intl. Conf. on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 279–299, 2016.

[81] Gorka Irazoqui, Mehmet Sinan Inci, Thomas Eisenbarth, and Berk Sunar. Wait a minute! a fast, cross-vm attack on aes. In Angelos Stavrou, Herbert Bos, and Georgios Portokalidis, editors, *Research in Attacks, Intrusions and Defenses*, pages 299–319, Cham, 2014. Springer International Publishing. ISBN 978-3-319-11379-1.

[82] Aditya S. Gangwar, Prathamesh N. Tanksale, Shirshendu Das, and Sudeepta Mishra. $\mathcal{F}lush + early\mathcal{R}$ELOAD: Covert channels attack on shared llc using mshr merging. In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6, 2024. doi: 10.23919/DATE58400.2024.10546609.

[83] Berk Gülmezoğlu, Mehmet Sinan undefinednci, Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. A faster and more realistic flush+reload attack on aes. In *Revised Selected Papers of the 6th International Workshop on Constructive Side-Channel Analysis and Secure Design - Volume 9064*, COSADE 2015, page 111–126, Berlin, Heidelberg, 2015. Springer-Verlag. ISBN 9783319214757. doi: 10.1007/978-3-319-21476-4_8. URL https://doi.org/10.1007/978-3-319-21476-4_8.

[84] Thomas Allan, Billy Bob Brumley, Katrina Falkner, Joop van de Pol, and Yuval Yarom. Amplifying side channels through performance degradation. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*, ACSAC '16, page 422–435, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450347716. doi: 10.1145/2991079.2991084. URL https://doi.org/10.1145/2991079.2991084.

[85] T. George Hornby. Side-channel attacks on everyday applications: Distinguishing inputs with f lush +r eload. 2016. URL https://api.semanticscholar.org/CorpusID:211266849.

[86] Mengjia Yan, Read Sprabery, Bhargava Gopireddy, Christopher Fletcher, Roy Campbell, and Josep Torrellas. Attack directories, not caches: Side channel attacks in a non-inclusive world. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 888–904, 2019. doi: 10.1109/SP.2019.00004.

[87] Zhen Hang Jiang and Yunsi Fei. A novel cache bank timing attack. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 139–146, 2017. doi: 10.1109/ICCAD.2017.8203771.

[88] David Lilburn Watson. Hey – get off my cloud! In Hamid Jahankhani, Ali G. Hessami, and Feng Hsu, editors, *Global Security, Safety, and Sustainability*, pages 224–232, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. ISBN 978-3-642-04062-7.

[89] Mengjia Yan, Read Sprabery, Bhargava Gopireddy, Christopher Fletcher, Roy Campbell, and Josep Torrellas. Attack directories, not caches: Side channel attacks in a non-inclusive world. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 888–904, 2019. doi: 10.1109/SP.2019.00004.

[90] Clémentine Maurice, Manuel Weber, Michael Schwarz, Lukas Giner, Daniel Gruss, Carlo Alberto Boano, Stefan Mangard, and Kay Römer. Hello from the Other Side: SSH over Robust Cache Covert Channels in the Cloud. In *NDSS*, volume 17, pages 8–11, 2017.

[91] Colin Percival. Cache missing for fun and profit. 2005. URL https://api.semanticscholar.org/CorpusID:14403674.

[92] Yunjing Xu, Michael Bailey, Farnam Jahanian, Kaustubh Joshi, Matti Hiltunen, and Richard Schlichting. An exploration of l2 cache covert channels in virtualized environments. In *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*, CCSW '11, page 29–40, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450310048. doi: 10.1145/2046660.2046670. URL https://doi.org/10.1145/2046660.2046670.

[93] Zhenyu Wu, Zhang Xu, and Haining Wang. Whispers in the hyper-space: high-speed covert channel attacks in the cloud. In *Proceedings of the 21st USENIX Conference on Security Symposium*, Security'12, page 9, USA, 2012. USENIX Association.

[94] Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. Cross processor cache attacks. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '16, page 353–364, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342339. doi: 10.1145/2897845.2897867. URL https://doi.org/10.1145/2897845.2897867.

[95] Samira Manabi Khan, Yingying Tian, and Daniel A. Jimenez. Sampling dead block prediction for last-level caches. In *Proceedings of the 2010 43rd Annual IEEE/ACM*

International Symposium on Microarchitecture, MICRO '43, page 175–186, USA, 2010. IEEE Computer Society. ISBN 9780769542997. doi: 10.1109/MICRO.2010.24. URL https://doi.org/10.1109/MICRO.2010.24.

[96] Ishwar S. Bhati, Huichu Liu, Jayesh Gaur, Kunal Korgaonkar, Sasikanth Manipatruni, Sreenivas Subramoney, Tanay Karnik, Hong Wang, and Ian A. Young. Write congestion aware bypass for non-volatile memory, last level cache (llc) dropping from write queue responsive to write queue being full and read queue threshold wherein the threshold is derived from latency of write to llc and main memory retrieval time, 2019.

[97] Nathan Gober, Gino Chacon, Lei Wang, Paul V. Gratz, Daniel A. Jimenez, Elvira Teran, Seth Pugsley, and Jinchun Kim. The championship simulator: Architectural simulation for education and competition, 2022.

[98] John L. Henning. Spec cpu2006 benchmark descriptions. *SIGARCH Comput. Archit. News*, 34(4):1–17, sep 2006. ISSN 0163-5964. doi: 10.1145/1186736.1186737. URL https://doi.org/10.1145/1186736.1186737.

[99] Scott Beamer, Krste Asanović, and David Patterson. The gap benchmark suite, 2017.

[100] Mario Werner, Thomas Unterluggauer, Lukas Giner, Michael Schwarz, Daniel Gruss, and Stefan Mangard. Scattercache: Thwarting cache attacks via cache set randomization. In *USENIX Security Symposium*, 2019. URL https://api.semanticscholar.org/CorpusID:199568473.

[101] Mostafa Hadizadeh, Elham Cheshmikhani, and Hossein Asadi. Stair: High reliable stt-mram aware multi-level i/o cache architecture by adaptive ecc allocation. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1484–1489, 2020. doi: 10.23919/DATE48585.2020.9116550.

[102] Elham Cheshmikhani, Hamed Farbeh, and Hossein Asadi. 3rset: Read disturbance rate reduction in stt-mram caches by selective tag comparison. *IEEE Transactions on Computers*, 71(6):1305–1319, 2022. doi: 10.1109/TC.2021.3082004.

[103] Sukarn Agarwal, Shounak Chakraborty, and Magnus Själander. Architecting selective refresh based multi-retention cache for heterogeneous system (armour). In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2023. doi: 10.1109/DAC56929.2023.10247878.

[104] M.K. Qureshi, D. Thompson, and Y.N. Patt. The v-way cache: demand-based associativity via global replacement. In *32nd International Symposium on Computer Architecture (ISCA'05)*, pages 544–555, 2005. doi: 10.1109/ISCA.2005.52.

[105] Daniel Sanchez and Christos Kozyrakis. The zcache: Decoupling ways and associativity. In *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 187–198, 2010. doi: 10.1109/MICRO.2010.20.

[106] A. Bhatla, Navneet, and B. Panda. The maya cache: A storage-efficient and secure fully-associative last-level cache. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pages 32–44, Los Alamitos, CA, USA, jul 2024. IEEE Computer Society. doi: 10.1109/ISCA59077.2024.00013. URL https://doi.ieeecomputersociety.org/10.1109/ISCA59077.2024.00013.

[107] Odilia Coi, Guillaume Patrigeon, Sophiane Senni, Lionel Torres, and Pascal Benoit. A novel sram-stt-mram hybrid cache implementation improving cache performance. In *NANOARCH: Nanoscale Architectures*, pages 39–44, July 2017. doi: 10.1109/NANOARCH.2017.8053704.

[108] Mehmet Kayaalp, Dmitry Ponomarev, Nael Abu-Ghazaleh, and Aamer Jaleel. A high-resolution side-channel attack on last-level cache. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2016. doi: 10.1145/2897937.2897962.

[109] Jaspinder Kaur and Shirshendu Das. Acpc: Covert channel attack on last level cache using dynamic cache partitioning. In *2023 24th International Symposium on Quality Electronic Design (ISQED)*, pages 1–8, 2023. doi: 10.1109/ISQED57927.2023.10129363.

[110] Jaspinder Kaur and Shirshendu Das. Tppd: Targeted pseudo partitioning based defence for cross-core covert channel attacks. *Journal of Systems Architecture*, 135:102805, 2023. ISSN 1383-7621. doi: https://doi.org/10.1016/j.sysarc.2022.102805. URL https://www.sciencedirect.com/science/article/pii/S1383762122002909.

[111] Jaspinder Kaur and Shirshendu Das. Rspp: Restricted static pseudo-partitioning for mitigation of cross-core covert channel attacks. *ACM Trans. Des. Autom. Electron. Syst.*, 29(2), jan 2024. ISSN 1084-4309. doi: 10.1145/3637222. URL https://doi.org/10.1145/3637222.

[112] James Bucek et al. Spec cpu2017: Next-generation compute benchmark. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, ICPE '18, page 41–42, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450356299.

[113] Zhenyu Sun, Xiuyuan Bi, Hai Li, Weng-Fai Wong, and Xiaochun Zhu. Stt-ram cache hierarchy with multiretention mtj designs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(6):1281–1293, 2014. doi: 10.1109/TVLSI.2013.2267754.

[114] Fangting Huang, Dan Feng, Wen Xia, Wen Zhou, Yucheng Zhang, Min Fu, Chuntao Jiang, and Yukun Zhou. Security rbsg: Protecting phase change memory with security-level adjustable dynamic mapping. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1081–1090, 2016. doi: 10.1109/IPDPS.2016.22.

[115] Fang ting Huang, Dan Feng, Wen Xia, Wen Zhou, Yu cheng Zhang, Min Fu, Chun tao Jiang, and Yu kun Zhou. Enhancing security of nvm-based main memory with dynamic feistel network mapping. *Frontiers of Information Technology Electronic Engineering*, 19(7):847–863, July 2018. ISSN 2095-9230. doi: 10.1631/FITEE.1601652. URL https://doi.org/10.1631/FITEE.1601652.

[116] Moinuddin K. Qureshi, Andre Seznec, Luis A. Lastras, and Michele 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*, pages 478–489, 2011. doi: 10.1109/HPCA.2011.5749753.

[117] Nitin Rathi, Swaroop Ghosh, Anirudh Iyengar, and Helia Naeimi. Data privacy in non-volatile cache: Challenges, attack models and solutions. In *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 348–353, 2016. doi: 10.1109/ASPDAC.2016.7428036.

[118] Mu-Tien Chang et al. Technology comparison for large last-level caches (l3cs): Low-leakage sram, low write-energy stt-ram, and refresh-optimized edram. In *IEEE HPCA*, pages 143–154, 2013.

[119] Chi-Keung Luk et al. Pin: building customized program analysis tools with dynamic instrumentation. In *ACM SIGPLAN PLDI*, PLDI '05, page 190–200, 2005. ISBN 1595930566.