Carlos Escuín Blasco

# Crafting Non-Volatile Memory (NVM) Hierarchies: Optimizing Performance, Reliability, and Energy Efficiency

Director/es

Ibáñez Marín, Pablo Enrique
Viñals Yufera, Víctor

ISSN 2254-7606

Universidad de Zaragoza
Servicio de Publicaciones

ISSN 2254-7606

Tesis Doctoral

# CRAFTING NON-VOLATILE MEMORY (NVM) HIERARCHIES: OPTIMIZING PERFORMANCE, RELIABILITY, AND ENERGY EFFICIENCY

Autor

Carlos Escuín Blasco

Director/es

Ibáñez Marín, Pablo Enrique
Viñals Yufera, Víctor

**UNIVERSIDAD DE ZARAGOZA**
**Escuela de Doctorado**

Programa de Doctorado en Ingeniería de Sistemas e Informática

2024

Universidad de Zaragoza
Escuela de Ingeniería y Arquitectura

# Crafting Non-Volatile Memory (NVM) Hierarchies: Optimizing Performance, Reliability, and Energy Efficiency

Ph.D. Thesis

Carlos Escuin Blasco
2023

Departamento de Informática e Ingeniería de Sistemas

Escuela de Ingeniería y Arquitectura

Universidad de Zaragoza

# Crafting Non-Volatile Memory (NVM) Hierarchies: Optimizing Performance, Reliability, and Energy Efficiency

Ph.D. Thesis

Author:

Carlos Escuin Blasco

Advisors:

Víctor Viñals Yúfera
Pablo Enrique Ibáñez Marín

2023

## Abstract

The escalating number of cores and accelerators in modern computing systems and the huge memory footprints and requirements of emerging applications beckon new challenges in the design of today memory hierarchies. One way to mitigate the impact of inefficient memory accesses resulting from these demanding memory requirements involves implementing larger on-chip cache memory hierarchies. The last-level cache (LLC), which is the last countermeasure when avoiding costly off-chip memory accesses, is traditionally built using SRAM technology; a technology that does not scale well in terms of area and static power. Emerging non-volatile memory (NVM) technologies have shown a great potential when replacing or augmenting conventional SRAM and DRAM memory structures such as the LLC, providing greater density and a reduced static power. However, these technologies suffer from energy-hungry write operations that, in turn, gradually degrade the materials eventually rendering the bitcells defective.

On the one hand, this dissertation studies and models the degradation of NVM bitcells due to write operations. Accurately analysing and assessing the interplay between such NVM degradation and the performance of the entire system is challenging. Therefore, we develop a forecasting procedure that comprehensively analyse the evolution over time of several figures of merit of the system; such as performance, lifetime, or energy. Besides, a trace-driven simulation tool is also developed in order to speed up the design space exploration of hybrid LLC architectures, insertion and replacement policies with the presence of defective NVM bitcells.

On the other hand, this dissertation unveils novel microarchitectural solutions to optimize such NVM-based LLCs for both performance and lifetime. These solutions consists of fault-tolerant NVM-based LLC designs that synergistically combine fine-grain disabling of defective memory regions, data compression, wear-leveling, and insertion/replacement policies. The proposed designs leverage data compression not only to reduce the bytes written to the NVM-based LLC but also to allow partially defective cache frames to hold compressed blocks. Moreover, the compression capabilities of the cache blocks are taken into account when guiding the insertion/replacement algorithms to further tune the lifetime and performance tradeoffs.

Computing-in-memory (CiM) paradigms deals with the inefficient memory accesses by bringing computational operations closer to memory structures, rather than the other way around, as in traditional von Neumann architectures. NVMs play a pivotal role in the CiM paradigm enabling analog computations within the memory array by exploiting their memristive properties. This dissertation also explores this emerging paradigm by revisiting an open-source CiM architecture, identifying and alleviating its limitations.

# Resumen

El incesante aumento del número de cores y aceleradores de los sistemas computacionales modernos y las exigentes necesidades de memoria de las aplicaciones emergentes plantean nuevos retos en el diseño de las actuales jerarquías de memoria. Una forma de mitigar el impacto de los ineficientes accesos a memoria que resultan de estos onerosos requisitos consiste en implementar jerarquías de memoria caché en el chip con más capacidad. La memoria caché de último nivel (LLC), que es la última contramedida para evitar los costosos accesos a memoria fuera del chip, se construye tradicionalmente con tecnología SRAM; una tecnología que no escacla bien en términos de área y potencia estática. Las tecnologías de memoria no volátiles (NVM) más recientes han demostrado tener un gran potencial a la hora de sustituir o complementar las estructuras convencionales de memoria SRAM y DRAM como la LLC, ya que proporcionan una mayor densidad y una potencia estática reducida. Sin embargo, estas tecnologías adolecen de unas operaciones de escritura que consumen mucha energía y que, a su vez, degradan paulatinamente los materiales, lo cual acaba por convertir las celdas en defectuosas.

Por un lado, esta tesis estudia y modela la degradación de las celdas NVM debida a las operaciones de escritura. Analyzar y evaluar con rigor la interacción entre esta degradación de las memorias NVM y el rendimiento de todo el sistema es todo un reto. Por lo tanto, desarrollamos un procedimiento de pronóstico que analiza de forma exhaustiva la evolución a lo largo del tiempo de varias figuras de interés del sistema como el rendimiento, el tiempo de vida útil de la LLC y la energía. Además, se ha desarrollado una herramienta de simulación basada en trazas de memoria para acelerar la exploración del espacio de diseño de arquitecturas de LLC híbridas, y de políticas de inserción y reemplazo para cachés con celdas NVM defectuosas.

Por otro lado, esta tesis presenta nuevas soluciones microarquitectónicas para optimizar dichas NVM-LLCs en términos tanto de rendimiento como de tiempo de vida útil. Estas soluciones consisten en diseños de LLCs tolerantes a fallos que combinan sinérgicamente la desactivación de regiones defectuosas de memoria, la compresión de datos, wear-leveling, y políticas de inserción y reemplazo. Los diseños propuestos aprovechan la compresión de datos no solo para reducir los bytes escritos en la LLC, sino también para permitir que los contenedores de caché parcialmente defectuosos puedan albergar bloques comprimidos. Además, la compresibilidad de los bloques de caché se tiene en cuenta a la hora de guiar a los mecanismos de inserción y reemplazo para afinar todavía más el equilibrio entre tiempo de vida útil de la LLC y el rendimiento del sistema.

La computación en memoria (CiM) aborda los accesos ineficientes a memoria acercando las operaciones de cómputo a las estructuras de memoria, en lugar de al revés, como en las arquitecturas von Neumann tradicionales. Las NVMs desempeñan un papel fundamental en el paradigma CiM, ya que permiten realizar cómputos de forma analógica dentro del array de memoria aprovechando sus propiedades resistivas. Esta tesis también explora este paradigma revisando una arquitectura CiM de código abierto, identificando y subsanando sus limitaciones.

# Publications

Part of this dissertation includes results already published or accepted for publication. The publications, in chronological order, are listed below:

1. **STT-RAM Memory Hierarchy Designs Aimed to Performance, Reliability and Energy Consumption.**

   *Carlos Escuin, Teresa Monreal, José María Llabería, Víctor Viñals, Pablo Ibáñez.*

   ACACES 2019 Poster Abstracts, pp. 231-234. July 2019. ISBN 978-88-905806-7-3. [42]

2. **HyCSim: A rapid design space exploration tool for emerging hybrid last-level caches.**

   *Carlos Escuin, Asif Ali Khan, Pablo Ibáñez, Teresa Monreal, Víctor Viñals, Jeronimo Castrillon.*

   System Engineering for constrained embedded systems (DroneSE and RAPIDO '22). New York, NY, USA: ACM, 2022, pp. 1–6. [41]

3. **L2C2: Last-Level Compressed-Cache NVM and a Procedure to Forecast Performance and Lifetime.**

   *Carlos Escuin, Pablo Ibáñez, Denis Navarro, Teresa Monreal, José María Llabería, Víctor Viñals.*

   Plos One, 18 (2), e0278346. DOI: `doi.org/10.1371/journal.pone.0278346`. [38]

4. **Compression-Aware and Performance-Efficient Insertion Policies for Long-Lasting Hybrid LLCs.**

   *Carlos Escuin, Asif Ali Khan, Pablo Ibáñez, Teresa Monreal, Jeronimo Castrillon, Víctor Viñals.*

   2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA), Montreal, QC, Canada, 2023, pp. 179-192, DOI: `doi.org/10.1109/HPCA56546.2023.10070968`. [39]

   ***HiPEAC Paper Award.***

5. **Leveraging Data Compression for Performance-Efficient and Long-Lasting NVM-based Last-Level Caches.**

   *Carlos Escuin, Asif Ali Khan, Pablo Ibañez, Teresa Monreal, Denis Navarro, José María Llabería, Jeronimo Castrillon, Víctor Viñals.*

   14th Annual Non-Volatile Memory Workshop. San Diego, CA. March 2023. URL: `http://nvmw.ucsd.edu/nvmw2023-program/nvmw2023-paper8-final_version_your_extended_abstract.pdf`. [40]

   ***Memorable Paper Award Finalist*.**

6. **MNEMOSENE++: Scalable Multi-Tile Design with Enhanced Buffering and VGSOT-MRAM based Compute-in-Memory Crossbar Array**

   *Carlos Escuin, Fernando García-Redondo, Mahdi Zahedi, Pablo Ibáñez, Teresa Monreal, Víctor Viñals, José María Llabería, James Myers, Julien Ryckaert, Dwaipayan Biswas and Francky Catthoor.*

   The 30th IEEE International Conference on Electronics, Circuits and Systems (ICECS). Istanbul, Turkey. December 2023. [37]

During the development of the thesis, open-source repositories have been published containing the code of the work carried out:

1. Forecasting procedure: https://gitlab.com/uz-gaz/l2c2-forecasting [36].

2. HyCSim: https://gitlab.com/uz-gaz/hycsim [35].

# Acknowledgements

No one succeeds without the help and support of others. Since *es de buen nacido ser agradecido*, the following lines are intended to express my gratitude to all the people whose efforts have made possible the accomplishment of this thesis.

First and foremost, to Pablo and Víctor for their continuous support, guidance, and constructive feedback during all these years. They passed on to me their passion for research. Their eagerness and persistence encouraged me even when things were not going as well as expected. To Teresa and J.M., for their kindness, critical view, and valuable feedback on the weekly meetings; because they have been the *second advisor*. To the people in the Grupo de Arquitectura de Computadores de Zaragoza (gaZ) for their warm hospitality and support, for making me feel like one of them since the first day. And to the gaZ Bees, for their comforting and distressing breaks, for our mutual understanding, for sharing this challenging journey.

To the people in Dresden, to Asif and Jeronimo, for opening the doors of the Chair for Compiler Construction. Getting to know and taking part in other ways of working is very enriching professionally; I am very grateful for that.

To the IMEC people, to Fernando, Dwaipayan, Mohit, Mahdi, and Francky. For making me take part in the ongoing projects of such an incredible environment and research center. To Fernando again, for your closeness, for your invaluable patience and dedication when introducing me to such a new and unknown topic for me.

To the Italian crew of Dresden, to the people of Kilimanjaro, and to the Ludovica FC. For being my family when I was abroad, friendships made overseas are always intense and special.

To all my close ones, for being a continuous inspiration, for always being there, for giving me hope, and for all the good times we spent and those that are yet to come.

Last but not least, to my grandparents and my parents for their unwavering support and for prioritizing and supporting the education I have always freely desired.

Carlos Escuin Blasco
Zaragoza, 22 de Noviembre, 2023

To my parents, who have unconditionally
supported me during my whole life.

# Contents

# List of Figures

# List of Tables

# Acronyms

**RAM** random access memory

**CPU** central processing unit

**DRAM** dynamic random access memory (RAM)

**LLC** last-level cache

**SRAM** static RAM

**NVM** non-volatile memory

**STT** spin-transfer torque

**SOT** spin-orbit torque

**MRAM** magnetic RAM

**ReRAM** resistive RAM

**PCM** phase change memory

**RTM** racetrack memory

**LRS** low resistance state

**HRS** high resistance state

**IoT** internet-of-things

**ECC** error correction code

**NV-LLC** non-volatile LLC

**L2C2** last-level compressed cache

**TDDB** time-dependent dielectric breakdown

**CiM** compute-in-memory

**CNN** convolutional neural network

**VGSOT** voltage-gate assisted SOT

**MTJ** magnetic tunnel junction

**RL** reference layer

**FL** free layer

**SL** source line

**WL** word line

**BL** bit line

**IMEC** Interuniversity Microelectronics Centre

**IPC** instructions per cycle

**BDI** Base-Delta-Immediate

**LRU** least recently used

**MRU** most recently used

**VLSI** very large-scale integration

**DCC** Decoupled Compressed Cache

**RDE** read disturbance error

**CE** compression encoding

**ECP** Error-Correcting Pointers

**SECDED** single error correction double error detection

|  |  |
|---|---|
| **DECTED** | double error correction triple error detection |
| **DRM** | Dynamically Replicated Memory |
| **OS** | operating system |
| **LCR** | low compression ratio |
| **HCR** | high compression ratio |
| **CB** | compressed block |
| **ECB** | extended CB |
| **RECB** | rearranged ECB |
| **FM** | fault map |
| **WLC** | wear-leveling counter |
| **FD** | frame disabling |
| **MPKI** | misses per kilo instructions |
| **APKI** | accesses per kilo instructions |
| **TAP** | Thrashing Aware Placement |
| **BH** | baseline hybrid |
| **BH_CP** | BH with compression |
| **CA** | compression-aware |
| **BW** | bytes written |
| **CA_RWR** | CA + read- write- reuse |
| **CP_SD** | compression with Set Dueling |
| **LB** | loop-block |
| **NLB** | non-loop-block |
| **RW** | remaining writes |

**WR** write rate

**CC** compression class

**GEMM** general matrix multiplication

**ISA** instruction-set architecture

**SIMD** single instruction multiple data

**RD** Row Data

**WD** Write Data

**IB** input buffer

**ADC** analog-to-digital converter

**DAC** digital-to-analog converter

**QoS** quality of service

**VM** virtual machine

**SLA** Service Level Agreement

**ML** machine learning

**LSTM** Long Short-Term Memory

**RNN** recurrent neural network

**DL** deep learning

**1D-CNN** one-dimensional CNN

**1D-pCNN** 1D layers with different dilation rates CNN

**RMSE** root-mean-square error

**FCFS** first-come, first-served

**NAS** Network-Architecture-Search

**VCMA** voltage-controlled magnetic anisotropy

| | |
|---|---|
| **KWS** | keyword spotting |
| **VWW** | visual wake words |

# Introduction

*You come at the king, you best not miss.*

Omar Little, The Wire.

# 1 Introduction

In the ever-evolving landscape of computing, the continuous pursuit of enhanced performance, computational capabilities, and energy efficiency has been fueled by groundbreaking principles such as Moore's Law and Dennard Scaling since the 60s. These pioneering postulates have steered the semiconductor industry towards achieving exponential growth in transistor density and processing power, enabling the development of smaller and more powerful electronic devices [30, 89]. However, the different manufacturing processes and integration capabilities resulted in performance improvements disparities between the central processing unit (CPU) and main memory dynamic RAM (DRAM) devices.

In the early 90s, Wulf et al. identified these improvement disparities between CPU and DRAM devices as exponential and predicted that the performance of future computing systems would be dominated by memory devices [128], which was called the *memory wall.* While CPU performance was improving 60% every year, DRAM was only doing so 10%, see Figure 1.1. This gap was expected to keep growing 50% every year [55, 98], posing a significant challenge to overall systems efficiency. However, in the last two decades, the deceleration of Moore's Law have slowed down the exponential growth of CPU performance [43, 111]. This, together with the relative improvements in DRAM, f.i. 3D stacking, turned this performance disparities not as abrupt as it was originally predicted. Far from having solved the problem, this deceleration of Moore's Law also made the number of transistors per-core that can be packed for the last-level cache (LLC) tends to be stagnated [43, 57, 111].

The wholesale generation of data together with emerging application domains such as machine learning, artificial intelligence, or bioinformatics, which have huge memory footprints, have only exacerbated the problem. Overcoming the memory wall challenge became an imperative goal for the research community, steering towards advancements in memory technologies, cache hierarchies, and memory management techniques to enable more efficient and balanced computing systems.

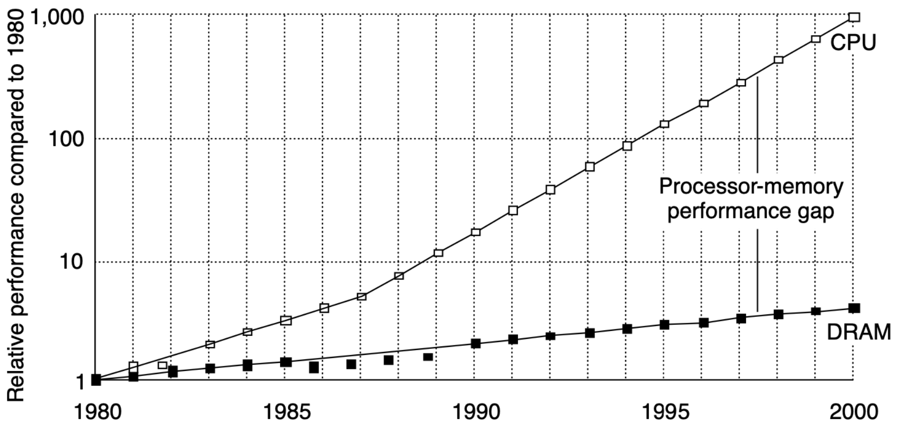


***Figure** 1.1: CPU - DRAM performance gap [55, 98].*

The memory wall problem has been tackled from various perspectives, encompassing the integration of emerging memory technologies and deeper cache memory hierarchies together with more sophisticated content management mechanisms, and the adoption of innovative computing paradigms that go beyond traditional von Neumann architectures.

On the one hand, one way to mitigate the impact of inefficient memory accesses resulting from the continuously growing memory requirements involves implementing larger on-chip cache memory hierarchies. More specifically, the shared LLC is the last countermeasure against costly off-chip memory accesses. As the number of cores/threads integrated on a chip outpaces the growth in bandwidth with main memory, it is thereby necessary to improve the LLC hit rate by not only increasing its overall size but also the size per core. Traditionally, the LLC implementation relies on static RAM (SRAM), a technology that does not scale well in terms of area and static power [43, 57, 74, 106]. Therefore, employing emerging non-volatile memory (NVM) technologies to replace or augment conventional LLCs is rising as a promising alternative.

In the last decade, several NVMs have made their way down to the memory hierarchy [62, 115]. Several NVM technologies such as spin-transfer torque (STT)- or spin-orbit torque (SOT)- magnetic RAM (MRAM), phase change memory (PCM), resistive RAM (ReRAM), and racetrack memory (RTM) can potentially replace or augment traditional SRAM or DRAM structures [101, 107, 118, 137]. In particular, they are attractive to build larger LLCs because they offer larger densities, lower static power and SRAM-competitive read latencies [5, 19, 44, 47, 71, 74, 118, 121, 124, 130]. Nonetheless, write endurance is still a hurdle so that these emerging technologies are deployed in large-scale manufacturing processes [24, 28, 44, 107, 117, 121, 123, 124]

On the other hand, beyond the challenges related to the memory wall, the data movement in von Neumann architectures impose significant overheads. The data transfer between DRAM and the CPU consumes two orders of magnitude more energy than a single floating point operation [82, 119]. Or even worse, a complete data fetch from main memory supposes more than three orders of magnitude more energy that a single integer addition [51]. To address this energy consumption disparities, emerging computing paradigms advocate for bringing computations closer to the memory, and processing data where it logically makes more sense. For instance, compute-in-memory (CiM) using NVMs is an emerging computing paradigm that exploits the *memristor* properties of NVMs to perform simple logic and arithmetic operations within the memory array. It allows to perform vector-vector and matrix-matrix multiplications, which are major operations in machine learning applications, in a very efficient way in terms of both performance and energy.

## 1.1 Non-volatile memory (NVM) technologies

NVM technologies such as PCM, ReRAM, RTM, or MRAMs provide better scalability tradeoffs compared to traditional SRAM and DRAM technologies. Contrary to DRAM and SRAM, writing a NVM cell involves altering physical state or property of the materials that compose such cells, which results in the definition of two resistive states: low resistance state (LRS), f.i. logical "1", and high resistance state (HRS), f.i. logical "0".

### 1.1.1 Magnetic RAM (MRAM)

Magnetic memories such as STT-MRAM relies on magnetic tunnel junction (MTJ) devices, which are composed by ferromagnetic (CoFeB) and dielectric (or insulating) (MgO) layers. Figure 1.2 depicts an STT-MRAM cell, consisting of an MTJ device and the access transistor. The MTJ device consists, in turn, of two ferromagnetic layers, the reference layer (RL) and the free layer (FL), separated by the dielectric layer. The magnetic orientation of RL is fixed, while the one of FL indicates the resistive state of the cell. If the magnetization of FL is programmed to be aligned with RL, the state is *parallel*, providing low resistance (LRS). If the magnetization is misaligned, the state is *anti-parallel*, providing high resistance (HRS). To write the cell, once the word line (WL) is set, a positive current from bit line (BL) to source line (SL) sets the MTJ device in parallel state, and vice versa, a positive current from SL to BL sets the MTJ device in anti-parallel state. To read the cell, WL is activated and a small voltage is applied in SL to sense the current in BL.

### 1.1.2 Phase change memory (PCM)

PCM relies on a chalcogenide alloy such as $Ge_2Sb_2Te_5$ (GST), see Figure 1.3a. This material is a kind of glass that is able to switch between two phases: *amorphous* and *crystalline*. The amorphous phase has high resistance (HRS) while the crystalline one has a lower one (LRS). The material can be changed between both phases by using current pulses that heat the material [103], see Figure 1.3b. The GST material is crystallized by injecting a long electrical pulse (SET pulse) that heats it above the crystallization temperature, but below the melting one. The long SET pulse gradually cools down the material making it possible the crystal growth. By contrast, a short but high electrical pulse (RESET pulse) is injected and then abruptly cut off to turn the GST into the amorphous phase. In this case,

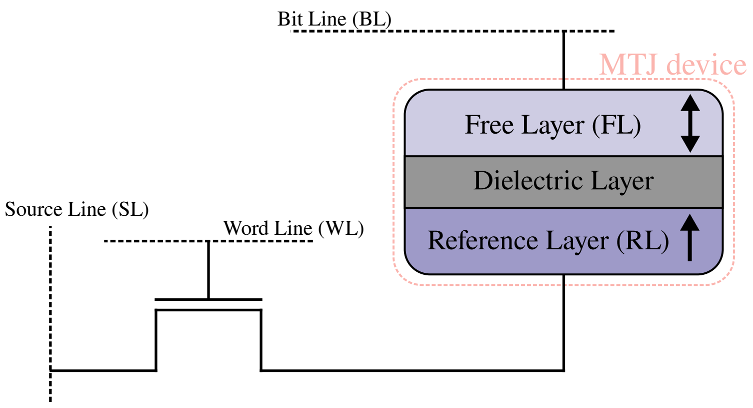
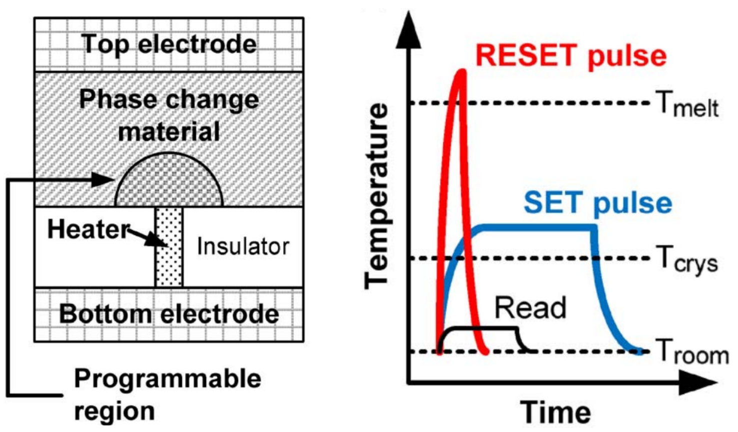


***Figure** 1.2: STT-MRAM cell. MTJ device together with the access transistor.*

the temperature exceeds the melting one and the material is suddenly cooled down into the amorphous state. In order to perform a read operation, the resistance of the cell is sensed out by means of a small and short pulse.



(a) PCM cell. (b) PCM cell switch and read diagram.

***Figure** 1.3: Schematic cross-section of a conventional PCM cell (a). Read, SET, and RESET pulses (b). [126].*

### 1.1.3 Resistive RAM (ReRAM)

ReRAM cells consist of a dielectric (or insulating) layer, either a metal oxide or a solid electrolyte semiconductor, placed between the top electrode and the bottom electrode, as shown in Figure 1.4. LRS is achieved by creating one or more *conductive filaments* in the dielectric layer, electrically connecting the top and the bottom electrodes. HRS is achieved by dissolving such filaments. The number, morphology and generation of these conductive filaments depend on the device materials, sizing and input stimuli [46].

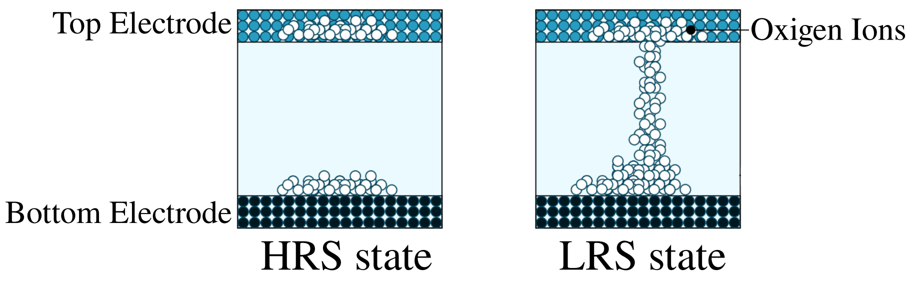


***Figure** 1.4: Cell diagram of a metal oxide based ReRAM cell [46].*

### 1.1.4 The endurance problem of NVM technologies

Writing 0 or 1 to an NVM bitcell requires to invest some energy for a period of time in order to alter the value of a physical property in one of the bitcell circuit materials, whose structure, components, dimensions and interface are critical to the proper functionality of the memory [88, 101, 107]. Write operations, besides being more costly in time and energy than read operations, eventually degrade bitcells, which render to lose its storage capacity. In this context, the *bitcell endurance is defined as the number of writes the bitcell will withstand before it breaks down and loses its storage capability.*

For example, in the case of STT-MRAM bitcells the wear produced by the cumulative effect of writes eventually leads to what is called time-dependent dielectric breakdown (TDDB). TDDB is the short-circuit of the thin dielectric layer, i.e. the dielectric layer in Figure 1.2, that isolates the two ferromagnetic layers, RL and FL: once the dielectric breakdown occurs the change is irreversible and the bitcell behaves as a small fixed-value resistor; it is no longer possible to distinguish between the parallel and anti-parallel states, whose respective resistances are designed to be sufficiently different to encode a bit reliably [14, 117].

In the case of PCM, the pulses that are injected into the GST material produce a thermal expansion and contraction that degrades the electrode storage contact, such that programming currents are no longer reliably injected into the cell. Eventually, stuck-at-1 or stuck-at-0 faults are produced because either the overheating of the GST material reduces the overal resistance of the cell or an induced defect in the memory cell leads to an open circuit [67, 78, 117].

In the case of ReRAM, the consecutive LRS-HRS state changes eventually produces the deficiency or excessive doping of oxygen vacancies (in the case of metal oxide ReRAM) in the cell, sticking the cell in either of the two states [117]. Some studies concluded that larger voltage amplitudes reduce the lifetime of these devices [46, 59].

#### Modelling the write endurance

The write endurance of each bitcell can be modeled as an independent random variable following a Gaussian distribution of mean $\mu = 10^k$ writes and coefficient of variation $cv = \frac{\sigma}{\mu}$, $k$ depending on the NVM technology and $cv$ usually taking values between 0.2 and 0.3 [28, 44, 60, 110, 112, 133]. The coefficient of variation reflects the variability in the manufacturing process. The endurance figures are different for each technology and depend on the manufacturer and the target market.

For instance, STT-MRAM endurance is subject to some design parameters tradeoffs such as retention time, area, power efficiency and read/write latency [48, 91]. It is therefore not surprising to find in the literature STT-MRAM endurance values from $10^6$ for embedded systems or internet-of-things (IoT) applications [24,48,79,124] up to $10^{12}$ for general purpose microprocessors [44, 58, 121].

## 1.2 Contributions

The contributions of this dissertation are multi-fold and can be classified in the different parts in which this thesis is divided:

**Part I. Microarchitectural enchancements for NVM-based LLCs.** Several works propose optimizations in order to enhance the performance of NVM-based LLCs. Others, focus on schemes that postpone or tolerate the occurrence of defective NVM bitcells, due to repeated write operations, in order to enlarge their lifetime. However, synergistically optimising for both performance and lifetime has rarely explored. Thus, the first part of the thesis tries to fill this gap, proposing microarchitectural solutions to optimize NVM-based LLCs for both performance and lifetime. The specific contributions are the following:

1. A detailed state-of-the-art review about mechanisms that either postpone or tolerate hard-faults. These studies include techniques that either decrease the number of writes, spread the writes among the whole memory structure, or tolerate (or recover from) a certain number of transient or permanent faults.

2. Last-level compressed cache (L2C2): A fault-tolerant non-volatile LLC (NV-LLC) design that combines fine-grain disabling of defective memory regions, data compression and wear-leveling in order to uphold high performance for longer.

    - It has the necessary metadata to separately identify faulty bytes in each cache frame[1] and a replacement mechanism that only considers those cache frames with sufficient capacity to accommodate the incoming compressed block.
    - It incorporates an intra-frame wear-leveling mechanism together with a block rearrangement circuitry that enables the spreading of write operations among the remaining non-faulty bytes within a frame.
    - Using VLSI synthesis, we showed that this block rearrangement logic is feasible in terms of latency, area and power for both reading and writing a block.

3. L2C2+N, the scaled version of L2C2. L2C2 design allows to seamlessly add an arbitrary number of *N* redundant bytes to each cache frame for the purpose of reducing the wear caused by writes from the beginning of operation. This thereby extends the time during which L2C2 provides near-peak performance.

[1]Hereinafter in this dissertation, the term cache *frame* designates the set of physical bitcells of the data array holding a cache *block*, either compressed or not.

4. A comprehensive analysis and evaluation in terms of performance and lifetime of the proposed mechanisms. For a fair comparison, state-of-the-art solutions are included in the evaluation.

Hybrid SRAM-NVM caches try to combine the best of both worlds, greater capacities and energy efficiency due to NVMs while absorbing harmful write operation by the SRAM part. State-of-the-art insertion policies identify and steer write-intensive blocks towards the SRAM part and read-intensive blocks towards the NVM part. These policies reduce the write traffic to the NVM part thereby enlarging the lifetime of such hybrid caches. However, these improvements are achieved by conservatively inserting blocks in the NVM part, which results in a limited performance. Our next contributions aim at bridging the gap between state-of-the-art performance and lifetime disparities by proposing novel hybrid LLC insertion policies that jointly optimize to provide both high performance and reasonable lifetime:

1. A detailed state-of-the-art review about hybrid LLCs, identifying the suboptimal performance of lifetime-oriented proposals. These studies include mechanisms that predict the behaviour of the blocks, steering only read-intensive (predicted) blocks towards the NVM part of the hybrid LLC.
2. Extend L2C2 into a hybrid approach, including a data array split into SRAM and NVM. In other words, provide a conventional hybrid LLC with the fault-tolerant mechanisms proposed for L2C2.
3. A novel insertion policy that considers not only the reuse properties of the incoming block but also the compressed size to steer the block towards the NVM or the SRAM part.
4. A threshold-based mechanism that tunes the write traffic to the NVM part. It allows more or less blocks to be steered towards the NVM part in order to balance performance and lifetime.
5. A comprehensive analysis and evaluation in terms of performance and lifetime of the proposed mechanisms. For a fair comparison, state-of-the-art solutions are included in the evaluation.

**Part II. Methodological improvements for NVM-based LLCs.** Properly assessing the feasibility and suitability of such microarchitectural enhancements is pivotal. However, the correct analysis and evaluation of the interaction between the degradation of such NVM structures and the performance of the system is challenging. The second part of this thesis propose methodological enhancements for the evaluation of such NVM-based LLCs that get their capacity lessened over time due to write operations. The specific contributions are the following:

1. A detailed state-of-the-art review on methodologies to assess lifetime and performance of NVMs. These methodologies include from analytical metrics to simple aging models. Besides, a state-of-the-art review on tools for design space exploration of hybrid LLCs. These tools include from cycle-accurate simulators to trace-driven ones.

2. Forecasting procedure. A procedure that estimates the evolution over time of several indexes of interest.
    - It combines consecutive simulations and predictions to forecast how the NVM bitcells become defective throughout time.
    - It supports different disabling granularities, data compression, fault-aware replacement mechanisms, and the aforementioned block rearrangement and wear-leveling mechanisms.
3. HyCSim. A trace-driven simulation tool for hybrid LLCs that allows for a rapid exploration and evaluation of different insertion policies.
    - A disabling manager that handles defective bitcells. It supports the entirely or partially disabling of cache frames due to hard faults.
    - Support for data compression mechanisms. The traces can provide the compressed block to be handled by the cache controller.
    - State-of-the-art hybrid insertion policies are implemented in the public version of the simulator.

**Part III. Compute-in-memory (CiM) using NVMs.** CiM using NVMs raised as an interesting computing paradigm to face the bottlenecks of traditional von Neumann architectures. A fair example of a CiM general purpose architecture is the MENMOSENE single-tile one [134, 135]. This architecture performs simple analog computations within the memory array and completes and delivers the result in the digital periphery. Besides, this architecture is provided with an instruction-set architecture (ISA) that bridges the gap between the high-level programming languages and the underlying circuit designs. The third part of this thesis extends the MNEMOSENE single-tile architecture focusing on the shortcomings of the original design. The specific contributions are as follows:

1. Revisit the internal buffering of the original MNEMOSENE single-tile architecture and enhance the bottleneck communication with the outer system.
2. Design a multi-tile architecture. This design includes a shared scratchpad memory and an interconnection framework to enable a seamlessly synchronization between tiles.
3. Comprehensive evaluation of different convolutional neural network (CNN) models mapped to the multi-tile architecture. This evaluation is performed for different NVM technologies.

## 1.3 Thesis structure

This dissertation is organised as follows. Part I presents the microarchitectural improvements for NVM-based LLCs. This part is divided, in turn, in two chapters. Chapter 2 introduces L2C2 and Chapter 3 presents the hybrid LLC insertion policies. Part II presents the methodological enhancements to the evaluation of NVM-based LLCs in terms of both performance and lifetime. This second part is divided, in turn, in two chapters as well.

Chapter 4 introduces the forecasting procedure and Chapter 5 introduces HyCSim. Finally, Part III explores the CiM paradigm using NVM technologies. In this regard, Chapter 6 introduces the enhancements to the MNEMOSENE CiM framework.

## 1.4 Thesis project framework

This thesis has been developed at "Grupo de Arquitectura de Computadores de la Universidad de Zaragoza (gaZ)", in the Departamento de Informática e Ingeniería de Sistemas (DIIS) and Instituto de Investigación en Ingeniería de Aragón (I3A).

The research work that encompasses this thesis has been partially funded by:

- The Aragon Government through a scholarship grant to fund a pre-doctoral contract through the University of Zaragoza.
- Project PID2019-105660RB-C21: "Jerarquía de memoria, gestión de tareas y optimización de aplicaciones", from the Agencia Estatal de Investigación and TIN2016-76635-C2-1-R: "Arquitectura y programación de computadores escalables de alto rendimiento y bajo consumo", from the Spanish Ministry of Economy and Competitive.
- The Aragon Government through the Research group recognition: T58_20R and T58_23R research group from Aragon Government and European Social Fund, and (3) 2014-2020 "Construyendo Europa desde Aragón" from European Regional Development Fund.

Two research internships have been conducted. The first one was a 3-months visit to the Chair for Compiler Construction at the Dresden University of Technology (Dresden, Germany) under the supervision of professor Jeronimo Castrillon. This internship was funded by a HiPEAC collaboration grant. The second one was a 6-months internship at Interuniversity Microelectronics Centre (IMEC) (Leuven, Belgium) under the supervision of Fernando García, Dwaipayan Biswas, and Francky Catthoor. This internship was partially funded by both the Aragon Government and IMEC.

# Part I

# Microarchitectural enchancements for NVM-based LLCs

*We're building something here, Detective, we're building it from scratch. All the pieces matter.*

Detective Lester Freamon, The Wire.

# 2
# L2C2: Last-level compressed non-volatile cache

*Several emerging NVMs are rising as interesting alternatives to build the LLC. Their advantages, compared to SRAM memory, are higher density and lower static power, but write operations wear out the bitcells to the point of eventually losing their storage capacity. In this context, this chapter introduces L2C2, a novel NV-LLC organization to extend the lifetime of the NVM data array. This design combines fault tolerance, data compression, and wear-leveling for the first time. Data compression is not used to store more blocks and increase the hit rate, but to reduce the write rate and increase the lifetime during which the cache supports near-peak performance. In addition, to support byte loss without performance drop, L2C2 inherently allows N redundant bytes to be added to each cache entry. Thus, L2C2+N, the endurance-scaled version of L2C2, allows balancing the cost of redundant capacity with the benefit of longer lifetime.*

*L2C2 has affordable hardware overheads compared to that of a baseline NV-LLC without data compression in terms of area, latency and energy consumption, and increases up to 6 – 37 times the time in which 50% of the effective capacity is degraded, depending on the variability of the manufacturing process. Compared to L2C2, L2C2+6, which adds 6 bytes of redundant capacity per entry, meaning 9.1% of storage overhead, can increase up to 1.4 – 4.3 times the time in which the system gets its initial peak performance degraded.*

## 2.1 Introduction

The goal of the cache subsystem in a shared memory multiprocessor is to reduce the number of main memory accesses. Specifically, the LLC filters requests from the lower-level caches turning slow main memory accesses into fast LLC hits, saving main memory bandwidth and power, and increasing system performance. However, the number of cores/threads integrated on a chip grows faster than the bandwidth with main memory. Therefore, it is necessary to improve the hit ratio of the LLC by increasing not only total size but also size per core/thread. Most LLCs are implemented with SRAM, a technology that does not scale well in terms of density and static power [106].

In the short to medium term, NVM technologies rise as an alternative to SRAM due to their higher density and lower static power. However, write operations on most NVMs cause noticeable wear on their bitcells, making their lifetime much shorter than that of SRAM. The simplest way to deal with an irreparable fault in a bitcell is to disable the memory region to which it belongs, with a size that depends on the context: a cache frame or a byte, or even a whole memory page if we consider main memory.

This chapter unveils the design and evaluation of *L2C2*, an NV-LLC design intended to operate with memory bitcells that will wear out with writes until they can no longer be reliably programmed. L2C2 is a new fault-tolerant last-level cache organization that relies on byte disabling, data compression, and an intra-frame wear-leveling to to extend the lifetime of degraded frames. In contrast to current alternatives, it is able to maintain high performance for a longer time, or in other words, for a given time of use it achieves higher performance, and it does so at a reasonable hardware cost. Moreover, its design is inherently scalable in terms of lifetime: simply adding N additional spare bytes to each frame, without modifying the design ideas, results in L2C2+N, the endurance-scaled version of L2C2, which is able to support the nominal capacity for longer.

On the one hand, the design of L2C2 carefully considers previous concepts of non-volatile main memory management and SRAM caches, namely:

- Support for byte disabling [45], by incorporating the necessary metadata to identify non-operational bytes. Besides, an error correction code (ECC) mechanism is incorporated with the ability to trigger an operating system routine that disables a byte by modifying such metadata.

- Base-Delta-Immediate (BDI) compression [99]. This data compression mechanism is selected because it provides high coverage and a good compression ratio. These two characteristics allow, simultaneously, to reduce the number of bitcells written (more duration) and to increase the possibilities of saving the block in frames of reduced size (more performance). In addition, its hardware implementation has low decompression latency.

- LRU-Fit replacement algorithm [45]. After appropriate experimentation, this option is selected. LRU-Fit is a locality-aware replacement algorithm, which selects the least recently used (LRU) victim cache frame among all those that are large enough to allocate the incoming compressed block (Fit).

On the other hand, L2C2 incorporates two original enhancements, which are crucial to maintain high performance for a longer time, namely:

- Intra-frame wear-leveling and compressed block rearrangement within the frame. We propose a new mechanism that achieves three key objectives: (a) wear out the live bytes of each frame evenly as the rest is failing, (b) upon inserting a compressed block into L2C2, rearrange the byte layout of the compressed block to write the appropriate subset of live bytes of the frame, and (c) the same but in the reverse direction, i.e., in case of an L2C2 hit, reconstruct the original layout of a compressed block, which is scattered in a partially broken frame, to supply it to the decompressor. Using very large-scale integration (VLSI) synthesis, that circuitry has been shown to be feasible in terms of area, latency and power consumption.
- Because the above mechanism is scalable, it is possible to add an arbitrary number of N redundant bytes to each frame, privately and without any change in the design. L2C2+N, the version of L2C2 with redundancy, thus has frames with 64+N data bytes that cooperate in storing compressed blocks from the beginning, extending the cache lifetime in proportion to the built-in N degree of redundancy.

The rest of the chapter is organised as follows. Section 2.2 reviews the literature and the state-of-the-art in NV-LLC proposals and mechanisms to deal with the wear-out problem. Section 2.3 lays the groundwork for NV-LLCs. Section 2.4 describes L2C2, showing the storage overhead, the detailed design of the block read and write hardware, and the latency penalty incurred in the block read service. Section 2.5 evaluates the degradation of L2C2 over time and compares it to various NV-LLC configurations. Finally, Section 2.6 concludes this chapter.

## 2.2 Related work

It is inherent to NVM technologies that writes deteriorate the memory bitcells. This is why NVM-based cache designs have mechanisms to 1) decrease the number of writes, 2) spread out the writes (wear-leveling), avoiding wearing hot spots, and 3) tolerate both transient and permanent faults. Thus, novel proposals for NV-LLC organizations focus on mechanisms to decrease and/or balance the number of writes, seeking to increase the lifetime and at the same time, if possible, counteract the high energy and latency cost of writes.

**Write reduction**. Several works propose to reduce the number of inserted cache blocks using some kind of filtering [4, 19, 105], or collaborating with the private levels [74]. Other techniques to reduce writes are closely tied to particular bitcell designs, supporting e.g., read-before-write [65], or early-write-termination [131, 139]. It is also worth mentioning the proposals for hybrid SRAM/NVM LLCs, which stand out for their great potential to reduce writes, in exchange for a more complex design that seeks to send as many write requests as possible to the SRAM part without losing performance or increasing power consumption [19, 27, 123].

**Wear-leveling mechanisms**. They focus on evenly distributing write operations throughout all the NV-LLC dimensions: cache sets, ways within sets, and bytes within frames [2,44,65,121]. These works seek to slow down write wear by avoiding the formation of hot spots, but unlike L2C2, none of them consider how to prolong service in the presence of faulty bitcells, nor do they seek to achieve as gradual a loss of performance as possible.

**Fault-tolerant mechanisms**. Any memory structure is subject to experience a bitcell failure during its operation, either transient or permanent. For example, STT-MRAM bitcells, in addition to being able to fail permanently due to write wear, they can also fail transiently in a number of different ways. From least to most important these transient failures in STT-MRAM memories are: *retention failure*, where the stored value changes without any read or write operation; *write failure*, in which a write operation does not change properly the written value; and *read disturbance error*, where a read operation switches the value originally stored, leaving a wrong value [21]. In NV-LLCs these transient errors can occur in both the tag and data arrays.

Several specific techniques have been proposed to mitigate transient errors [20,22,23]. These techniques are orthogonal to our proposal since they deal with healthy bitcells. They could therefore be integrated into L2C2, which seeks, in a complementary way, to maintain the population of healthy bitcells as large as possible and for as long as possible.

To avoid a system crash, regardless of the transient or permanent nature of the error, a dedicated hardware must detect the error and correct it. To achieve this, fault-tolerant caches must protect each tag and cache frame with an ECC mechanism, capable at least of single error correction double error detection (SECDED), and often capable of coping with double error correction triple error detection (DECTED). For example, Wu et al. to mitigate read disturbance errors in STT-MRAM LLCs, propose to dynamically switch between SECDED to DECTED, and vice-versa, according to a temperature threshold for individual cache banks [127]. As a result, the devoted ECC code storage changes according to thermal stress.

Redundancy can be included in the error correction code itself, allowing to correct N errors instead of just one [72,110]. However, the overhead required by such ECCs increases rapidly with N, to the point of making it impractical.

Besides, if permanent errors accumulate in several bitcells of the same frame, in the end no solution based on ECCs is scalable, since after a certain number of errors it will not be possible to recover the correct value. The simplest solution is *frame disabling*, already present in commercial processors long time ago [17,129]. It consists of disabling the entire cache frame as soon as the error detection limit is reached, since one more permanent error could not be processed. In contrast, L2C2 relies on finer control of the disabling granularity, allowing to disable individual bytes in each frame and thus, together with block compression, increasing the cache lifetime.

Alternatively, redundancy can be added outside the ECC mechanism by noting permanently failed bitcells and correcting their value [110,112,133]. For example, Schechter et al., in the context of main memory proposes the Error-Correcting Pointers (ECP) mechanism that stores for each faulty bitcell its frame position and the value it should store, e.g. a nine-bit pointer for a 64-byte memory frame and a one-bit data, respectively [110]. The extra storage cost limits this approach to a moderate number of faulty cells. In fact, the authors evaluate the mechanism for up to N = 6 defective bitcells (ECP-6).

Other work proposes to take advantage of memory frames with defects without having to disable them entirely. For example, Ipek et al. proposes Dynamically Replicated Memory (DRM), that stores a memory page in two partially faulty page frames [60]. Or, with a higher complexity, Jadidi et al. advocate the use of compression to harden main memory [63]. They assume a PCM with ECP-6 protection for each 64-byte frame. Their mechanism allows storing a compressed block in a degraded frame, as long as there is a contiguous chunk within the frame, called compression window, of size greater than or equal to the compressed block, and with no more than 6 bitcell faults. This allows a memory frame to be used even if it has more than 6 faults, as long as they are outside the compression window. In summary, this proposal increases memory lifetime by three aggregate effects: it has a repair mechanism, it decreases the write rate by the same amount as the compression rate achieved, and it does not create write hot spots because it has an intra-frame wear-leveling mechanism. However, although its ideas are inspiring, this proposal has been developed to collaborate with operating system (OS) paging system and its direct transfer to cache memory hardware is not straightforward at all.

Finally, the possibility of storing compressed blocks in NV-LLCs has hardly been explored and, anyway, has never been proposed to extend lifetime. For example, Choi et al. explores an adaptation of the Decoupled Compressed Cache (DCC) compression scheme proposed for SRAM caches [108], but applying it to embedded NVM caches [26]. Similar to the DCC scheme for conventional caches, the aim is to increase the effective capacity by allowing the total number of compressed blocks stored in a cache set to exceed the nominal associativity. Using a set dueling mechanism, they dynamically adjust the activation/deactivation of compression to balance the miss rate vs. write rate tradeoff, concluding that their proposal increases energy efficiency, but decreases lifetime by 8% with respect to a cache without compression.

Mittal proposes a technique called SHIELD that uses compression to mitigate the effects of read disturbance errors (RDEs) in STT-MRAM [87]. The approach is to process misses by inserting *two* identical copies of the same compressed block in the target cache frame. For this purpose, SHIELD uses, like L2C2, a BDI compression scheme [99]. The first read leaves one of the two copies unusable by the RDEs, but a second copy is still intact for a second service. In this way, often, cache block reads do not require a restore (write-after-read), costly in energy and cache bank occupation [122].

Data compression has also been proposed in the context of caches operating at near-threshold voltage. Ferrerón et al. propose the Concertina cache, which provides each frame with a bit vector or a few pointers identifying the bytes that fault when the supply voltage drops [45]. These metadata are calculated once, by scanning the cache when entering in low-voltage mode, and do not change as long as the supply voltage remains constant. Before inserting a new block, a simple null subblock compression mechanism searches in LRU order for the existence of a frame with enough live bytes. Concertina does not need or seek to level write wear, nor requires a high-coverage compression mechanism, but part of its design will be useful for the operation of L2C2.

## 2.3 Background

This section briefly reviews the background regarding data compression in the context of NVMs, with emphasis on BDI compression, and the addition of redundant capacity.

### 2.3.1 Data compression

Data compression reduces the block size. This is beneficial in the NVM context because it allows fewer bits to be written and, consequently, compression has the potential to extend the lifetime of the main memory or cache [26, 31, 63, 96], and/or to decrease the RDE rate [87]. Yet, compression has another benefit in the context of a byte-level fault tolerant NV-LLC such as L2C2: it allows cache frames with dead bytes to hold blocks if compression is high enough [45, 63]. Any compression mechanism that achieves wide coverage even at the cost of a moderate compression ratio can be useful, so that a large percentage of blocks, once compressed, can be stored in degraded cache frames. On the other hand, the decompression latency must be very low in terms of processor cycles, since decompression is on the critical path of the block service and may affect system performance.

The chosen mechanism is *Base-Delta-Immediate* (BDI), as it achieves high coverage, fast decompression (1 cycle) and a substantial compression ratio [99]. BDI is based on value locality, i.e. on the similarity between the values stored within a block. It assumes that a 64-byte block is a set of fixed-size values, either 8 8-byte values, 16 4-byte values, or 32 2-byte values. It determines whether the values can be represented more compactly as a *Base* value and a series of arithmetic differences (*Deltas*) with respect to that base.

A block can be compressed with several Base + Delta combinations which are computed in parallel. An example with 14 BDI compression encodings (CEs) is shown in Table 2.1[1], along with the size values for the Base, Delta and the total compressed size. Thus, the compression mechanism chooses for each block the compression encoding (Base + Delta combination) that achieves the highest compression ratio.

***Table** 2.1: BDI CEs and their sizes, in bytes.*

| Name | Base | Delta | Size | Name | Base | Delta | Size |
|---|---|---|---|---|---|---|---|
| All Zeros | 0 | 0 | 0 | B2Δ1 | 2 | 1 | 37 |
| Rep. V(8) | 8 | 0 | 8 | B8Δ4 | 8 | 4 | 37 |
| B8Δ1 | 8 | 1 | 16 | *B8Δ5 | 8 | 5 | 44 |
| B4Δ1 | 4 | 1 | 21 | *B4Δ3 | 4 | 3 | 51 |
| B8Δ2 | 8 | 2 | 23 | *B8Δ6 | 8 | 6 | 51 |
| B8Δ3 | 8 | 3 | 30 | *B8Δ7 | 8 | 7 | 58 |
| B4Δ2 | 4 | 2 | 36 | Uncomp. | – | – | 64 |

[1]To calculate the sizes in this example, an *implicit* Base, taking the value zero, is assumed. This allows both very large and very small values to be coded at the same time but adds one bit per value to distinguish whether to consider the calculated or the implicit Base.

### 2.3.2 Addition of redundant capacity

The reliability of the NV-LLC can be improved by adding redundant capacity. This can be done by using classical ECCs or more sophisticated techniques [60, 110, 112, 127, 133]. The maximum number of bit errors that can be detected and corrected is limited by the available area and energy budget. For instance, Schechter et al. propose ECP, an ECC mechanism that encodes the location of defective bitcells and assigns healthy ones to replace them [110].

However, in order to further increase reliability, a substantial portion of the redundant capacity could be dedicated to the replacement or expansion of the rated cache capacity stated in the commercial specification. Both alternatives will be evaluated later in this chapter.

## 2.4 Fault-tolerant NV-LLC microarchitecture

In this section, we present an overview of the L2C2 organization, with emphasis on BDI compression adaptation and metadata layout. In addition, we explain and analyze in detail the block rearrangement and replacement mechanisms. Finally, we explore how to add redundant capacity to improve reliability.

### 2.4.1 General overview

#### NVM-friendly non-inclusive LLC

Non-inclusive LLC designs increase caching capacity by only partially duplicating data between the private and shared levels. The non-inclusive relationship allows replacing a block in LLC without having to invalidate copies in the private levels [136]. In an NVM-friendly implementation of this model, a miss in all cache levels involves a main memory access that takes the block directly to the private L1/L2 levels. In turn, the victim block replaced in L2, clean or dirty, is sent to the LLC and written if it was not there [29]. Most NVM-based LLCs follow this mostly-exclusive implementation because it reduces the write traffic in the LLC [18, 19, 84, 105].

Therefore, a non-inclusive organization is used to minimize writes in L2C2, see Figure 2.1. A block is inserted into L2C2 by effect of a replacement in L2, provided that the block was not already in L2C2. In case of a write miss (GetX) in L1 and L2, and a hit in L2C2, the corresponding block is brought to L1/L2 and invalidated in L2C2. This immediate invalidation improves LLC performance because it leaves room for the replacement algorithm to reuse it as needed. The obsolete copy of the invalidated block in LLC will be written anyway when the dirty block is evicted from L1/L2. These coherency features are already implemented in the MOESI_CMP_directory Ruby protocol in gem5 [83].

#### Replacement algorithm

To select the victim block, L2C2 takes into account the recency order according to the following rules: 1) inserted blocks are placed in an LRU list at the most recently used (MRU) position (lowest replacement priority), 2) a read hit in L2C2 places the block to the MRU

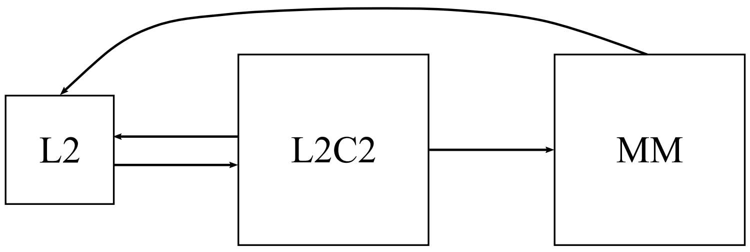


***Figure** 2.1: Block flow diagram of the non-inclusive model.*

position, and 3) replacement of a clean block in the private caches is communicated to L2C2; in case such a block is present, it is also placed at the MRU position.

However, if the LRU cache frame does not have sufficient capacity for the incoming compressed block, it cannot be used as a victim. Then there are two possibilities, either to search in order from least to most recent for the first frame with sufficient capacity (LRU-Fit policy) or to choose the frame with the smallest possible capacity, and if there are several with the same capacity, the LRU one (LRU-Best-Fit policy). Ferrerón et al. test both alternatives and choose LRU-Fit for its better performance [45], but since in their context writes do not produce degradation, the LRU-Best-Fit policy could be advantageous for the L2C2 design. LRU-Best-Fit avoids writes on the highest capacity frames, and therefore poorly compressible blocks would see their residency opportunities increase. Therefore, in Section 2.5.6 the two policies will be confronted.

#### Bitcell fault detection

Memory cells become defective after a certain number of writes. It is thereby essential to handle these permanent faults without losing information. From the architectural perspective, these memory structures must be provided with ECCs to detect and correct hard faults. We assume Hamming SECDED protection in all arrays. In particular, we use code (527, 516) for the NVM data array: it can correct one fault and detect up to two faults. Besides, this ECC mechanism, upon detecting and correcting a single bit fault, triggers an OS exception, notifying the identity of the faulty byte [133]. In order to prevent a second uncorrectable error from arising within the same region, the exception routine will disable the corresponding region, a whole frame using frame disabling, or a byte in L2C2. Note that this ECC protection does not bring any additional overhead as they already exists in SRAM LLCs to cope with soft and transient faults [3, 9, 44, 66, 116, 132]; for instance, AMD Zen's SRAM LLC employs DECTED protection [116].

Different disabling granularities in the LLC have different performance implications. For example, disabling at frame granularity incurs little overhead but severe degradation of capacity and, thus, performance. Conversely, disabling at a finer granularity, such as at byte level, requires more metadata (overhead) but allows live bytes within a frame to be used [45, 120, 125]. By leveraging compression, these partially disabled frames can be used as functional frames, and the impact of bitcell failures on performance can be effectively mitigated. For instance, if a byte in an NVM cache frame (64B) is disabled, it can still be used to store all cache blocks of compression encodings B8$\Delta$7 and above ($\leq$ 58B), see Table 2.1.

#### Wear-leveling mechanism

Writing compressed blocks in a frame is a new source of imbalance in the wear of the cells acting within the frame itself. As we will quantify, if, for example, compressed blocks are always stored from the beginning of the frame, the first bytes of the frame will receive more writes than the last ones.

Therefore, an intra-frame wear-leveling mechanism is needed to evenly distribute the writes within the frame. We assume a global counter modulo the cache frame size [63]. Blocks are written into the frames starting from the byte indicated by this counter and using the frame as a circular buffer. Each time the value of the counter is changed, the entire cache must be flushed, but since this must be done every few days or weeks, the impact on performance is negligible. Further details on how this mechanism copes with partially defective frames can be found in Section 2.4.6.

### 2.4.2 BDI adaptation

Pekhimenko et al. focus their application on achieving a large average compression ratio and therefore dispense with compression encodings with small compression ratios [99], those marked with an * in Table 2.1. However, L2C2 incorporates them, because in this way frames with few defective bytes will be able to store low compression blocks and thus performance increases noticeably [45].

To quantify the importance of such low compression blocks, Figure 2.2 shows a classification of all blocks written in L2C2 according to the achieved BDI compression ratio for the SPEC CPU 2006 and 2017 applications used in this work. On average, 22% of the blocks written are uncompressible (Unc), 29% have low compression ratio (LCR) (compressed block size $> 37$) and 49% have high compression ratio (HCR) (compressed block size $\leq 37$). For instance, if all frames in an L2C2 cache have a faulty byte, and the compression mechanism does not use the LCR encodings, the chance to store 29% of the blocks would be lost.

### 2.4.3 L2C2 metadata

The tag array undergoes the most write requests as it must keep the coherence and replacement states up to date. Should these bitcells fail, the entire data frame should be deactivated. Therefore, we assume the tag array is built using SRAM technology, free of wear by writing. Our proposal only adds a 4-bit field to store the frame capacity to each tag array entry. This frame capacity is represented in terms of the largest compression encoding the frame can allocate, see Figure 2.3. The data array is built using NVM technology. Each frame must have a capacity of 66 bytes: 64 data bytes plus one or two metadata bytes: up to 11 ECC bits and 4 bits representing CE of the data block.

In addition, a fault map is needed along with the data array to identify faulty bytes. Every frame has an associated fault map that points out the faulty bytes. This fault map information is initialized to '1's indicating that all bytes in a frame are non-defective. It requires 66 bits for each frame and is updated every time a byte becomes faulty, i.e., at most 66 times (until the frame is completely dead). This low amount of write accesses leads to no wear problems, and thus the fault map can be implemented in NVM technology.

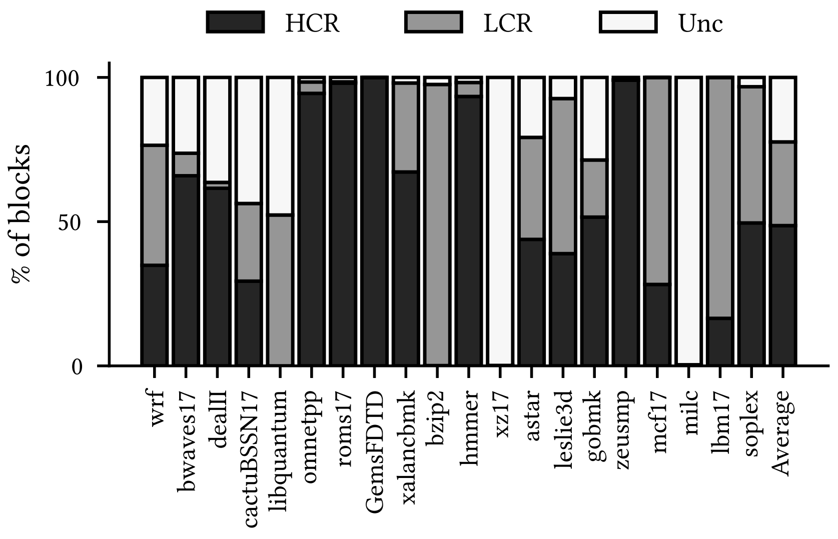


***Figure** 2.2: Block classification regarding its compression ratio for the selected SPEC CPU 2006 and 2017 applications. LCR blocks correspond to the CEs marked as * in Table 2.1.*

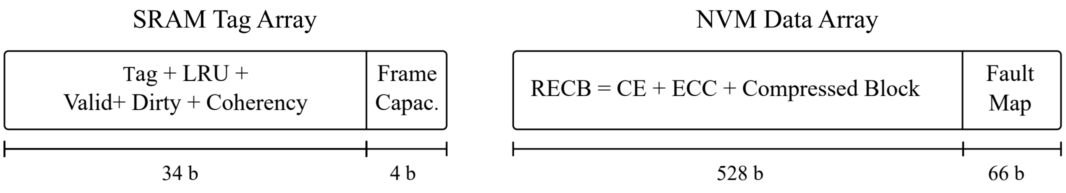


***Figure** 2.3: Layout of a frame entry in the SRAM tag and NVM data arrays.*

### 2.4.4 Block writing

Figure 2.4 shows the flow of writing a block into L2C2. First, the BDI compression units receive every incoming block B (64 B) [① BDI compression]. The result of each compression unit is a) whether the block is compressible or not and, if so, b) the compressed block (CB). As a result, CB with the highest compression ratio (CB, 0-64 B) is selected and the corresponding CE (4 b) is obtained.

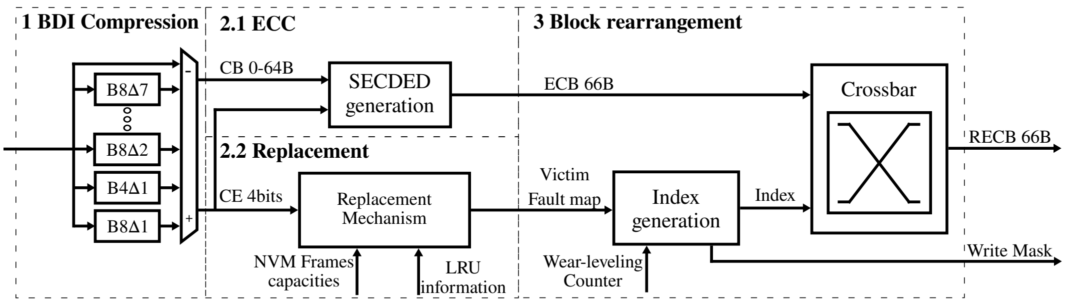


***Figure** 2.4: Block writing flow.*

The extended CB (ECB) is then formed by combining CB with the 4-bit CE and the 11-bit SECDED code. The ECCs are computed from 516-bit, i.e., the combined CE (4-bit) and 512-bit vector (the CB bit vector plus the required number of zeros to make 512-bit)

[2.1 ECC]. In parallel to SECDED generation, the replacement algorithm will look for the target frame among those with an effective capacity greater than or equal to the incoming CB (Fit-LRU) [45] [2.2 Replacement]. For this, the replacement logic considers the CE of the incoming block B along with the capacities and LRU order of the frames still alive in the involved cache set.

Besides, L2C2 is also provided with a block rearrangement circuitry that scatters the ECB throughout the non-faulty bytes of the target frame, generating the rearranged ECB (RECB) and a write mask for selective writing [③ Block rearrangement]. This block reordering synergistically works with an intra-frame wear-leveling mechanism to evenly distribute the wear of write operations across the remaining non-faulty bytes in the target frame. To do this, the block rearrangement circuitry maintains a counter that indicates the byte at which the write operation is performed. This counter is global, shared among all sets, and increments after long periods of time (a few hours or even days) so that the writing region of the frames gets shifted over time [63].

### 2.4.5 Block reading

Analogously, but in the opposite order to the writing flow, Figure 2.5 summarises the reading flow of a block in L2C2. First, RECB is read. In parallel, the index vector I[i] is computed as for writing the block. This index vector is now used to obtain ECB from RECB. First, the block is rearranged using as input RECB, the fault map, and the wear-leveling counter [① Block rearrangement]. Then, the ECC of ECB is checked [② ECC], and CB is eventually decompressed so that B is obtained and forwarded to L2/L1 [③ BDI Decompression].

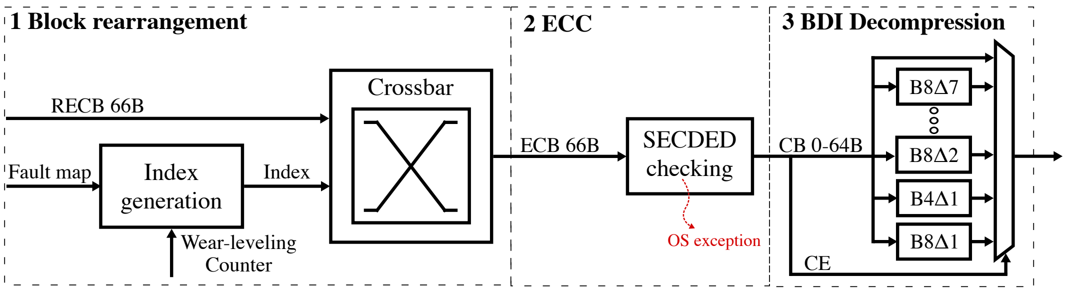
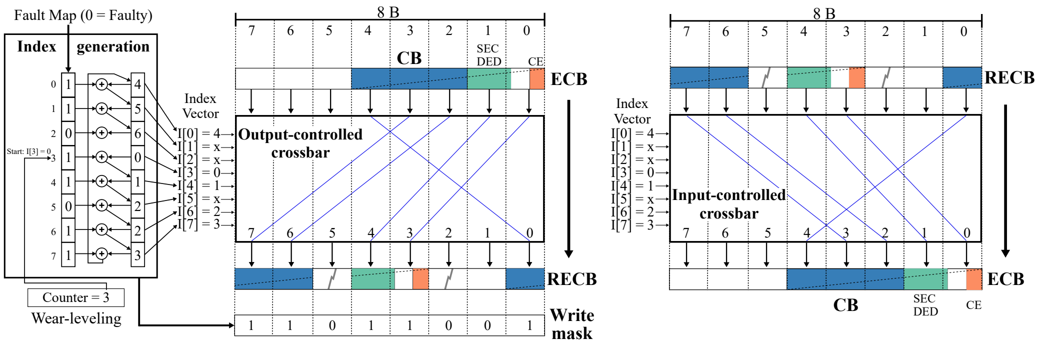


*Figure 2.5: Block reading flow.*

### 2.4.6 Rearrangement logic

The rearrangement logic is composed of two elements: index generator and crossbar. The index generator determines the mapping from ECB bytes to RECB bytes (L2C2 write) or conversely, from RECB bytes to ECB bytes (L2C2 read) while the crossbar transfer the bytes from the input ports to the corresponding output ports.

Figure 2.6a shows an example of rearranging a 5-byte ECB for scattering into an 8-byte frame with faulty bytes (2 and 5) while Figure 2.6b shows the opposite, how a 5-byte RECB is gathered from the same 8-byte frame. Focusing on the write example, on the left side of the Figure 2.6a, the index generator computes an index vector I[i] from the fault map and the wear-leveling counter. Each index indicates which byte of the ECB is to be placed in each RECB byte (x stands for don't care). For example, I[6]=2 indicates that byte 2 of the ECB is placed in RECB byte 6. Thus, this index vector controls the output ports of the crossbar used to obtain the RECB. In the example, crossbar output 6 selects input 2. For the write mask, the first *n* positions starting from the wear-leveling counter value, and corresponding to non-faulty bytes, are set to 1s; *n* being the ECB size.



(a) Writing example. (b) Reading example.

***Figure** 2.6: Example of a 5-byte block rearrangement for writing (a) and reading (b).*

Figure 2.6b shows a rearrangement example of the same 5-byte RECB from Figure 2.6a for delivery to L2. In this case, each index indicates the target output crossbar for each input, e.g., I[6]=2 means that input 6 is forwarded to output 2. Within the already aligned block (ECB), CE indicates the length of the compressed block. This value is employed to fill the bytes not used to store CB with zeros in order to match the SECDED previously generated during the writing. Besides, it selects the corresponding BDI decompressor.

Algorithm 1 describes the index generation for both writing and reading. It takes, as inputs, the fault map (FM) of the target frame, the wear-leveling counter (WLC), and the size of the block (extracted from CE). The outputs are the index vector I[N = *frame size*], that controls the output ports of the crossbar, and the write mask (WM) in the case of writing.

The first for loop (line 2) computes indexes without considering WLC. That is, assuming that, for example, ECB is to be rearranged starting with the byte zero of the destination frame. Note that the calculation of each iteration uses the result of the previous one. This implies using $N$ adders in series. Alternatively, our implementation uses a tree of adders, which reduces the computation time to that of $log_2(N)$ adders in series. Each adder uses $log_2(N)$ bits at most.

In the two next loops (lines 5 and 6), the indexes are adjusted regarding WLC. Now, the iterations within each loop are independent and can be computed in parallel. The computation time of the two iterations is, thus, the equivalent of two adders in series.

Finally, the last loop (line 7) computes the write mask. This loop can be synthesized with an array of 64 7-bit comparators. These comparators act on the computed indexes and their operation can overlap with the crossbar traversal.

The index vector is computed when writing and reading in the same way. In the write circuit, the crossbar is an array of multiplexers governed directly by the index vector. In the read circuit, the crossbar acts as right-aligner and is more complex. Our implementation

**Algorithm 1:** Index generation.

```
Input:
FM: N-bit vector fault map
WLC: wear-leveling counter
size: ECB size, computed from CE
    $0 \leq WLC, size \leq N - 1$
Output:
I[N]: N crossbar output port indexes
WM[N]: write mask N-bit vector

I[0] = 0
for i=1; i<N; i++ do  I[i] = I[i-1] + FM[i-1];
T = I[N-1] + FM[N-1];
WLCI = I[WLC];
for i=0; i<N; i++ do  I[i] = I[i] - WLCI;
for i=0; i<WLC; i++ do  I[i] = I[i] + T;
for i=0; i<N; i++ do
    if I[i] < size && FM[i] == 1 then
        WM[i] = 1
    else
        WM[i] = 0
```

assumes $N \times N$ comparators of $log_2(N)$ bits and $N$ output multiplexers of $N$ bytes to 1 byte with decoded control. The decoded control of the multiplexer that produces the byte $i$ is generated by $N$ comparators between the value $i$ and the $N$ elements of the index vector.

### 2.4.7 VLSI implementation

In order to put the costs and delays of the rearrangement logic into context, we select a L2C2 built with 22nm STT-MRAM technology, the largest scale of integration available in NVSim [33]. Table 2.2 shows area, latency and power of the SRAM tag array and the STT-MRAM data array, which make up the 4MB cache banks used in the experimental section.

Both writing and reading rearrangement logic are outside the L2C2 core, but the latter is located in the critical path of block delivery to L1/L2. In order to quantify their physical features both have been specified, simulated and laid out with the Synopsys Design Compiler R-2020.09-SP2 and Synopsys IC Compiler R-2020.09-SP2. Due to the lack of a 22nm library, we used the SAED16nm FinFET Low-Vt technology in worst case condition (typical-typical, 125 °C and 0.8 volts). These tools allowed us to estimate post-layout costs in terms of area, latency and power consumption. The dynamic power values were calculated from the cache activity factors measured during the workload simulations. The latency of the RECB → ECB logic (0.38 ns) plus the delay and setup times of the input and output registers, respectively, can be estimated at about two cycles at 3.5 GHz. That is, rearranging and

*Table 2.2: Hardware cost comparison.*

| | SRAM Tag Array 22 nm | STT-MRAM 4MB Data Array 22 nm | ECB→RECB 16 nm | RECB→ECB 16 nm |
|---|---|---|---|---|
| Area (mm$^2$) | 0.233 | 1.222 | 0.021 | 0.025 |
| Latency (ns) | 0.44 | 2.66 | 0.33 | 0.38 |
| Dynamic read power (mW) | 0.11 | 6.05 | – | 0.53 |
| Dynamic write power (mW) | 0.11 | 17.41 | 0.73 | – |
| Static power (mW) | 5.69 | 9.62 | 0.53 | 0.7 |

decompression increases the L2C2 load-use latency, with respect to a frame-disabling cache, from 30 to 32 cycles, a 6.7%.

In summary, looking at the figures as a whole, the overhead seems to be affordable on all metrics. Regarding storage costs, Table 2.3 also provides a comparison between all the evaluated cache candidates.

### 2.4.8 L2C2+N: adding redundant capacity to L2C2

Providing L2C2 with a few spare bytes in each frame could be very convenient since it would allow to continue working without loss of performance after the failure of several bytes of each cache frame.

The design presented so far allows to add N spare bytes in a very straightforward way: just increase each frame from 66 to 66+N bytes in the data array, and also increase the fault maps from 66 to 66+N bits. In addition, the rearrangement logic has to be extended to handle 66+N byte blocks, and the wear-leveling counter has to count modulo 66+N. Without further changes, the wear-leveling logic will take care of distributing the writes throughout the available 66+N bytes. A frame will only start to impose performance constraints when its effective capacity falls below 66 bytes.

## 2.5 Evaluation

This section shows the evolution of capacity and performance for several NV-LLC organizations, from 100% to 50% capacity, along with experiments on wear-leveling, replacement, cache size and workload. Four NV-LLCs candidates are analyzed, two based on frame disabling and two on byte disabling plus compression:

- **Frame disabling (FD) cache**. A bitcell failure is just handled by disabling the corresponding frame [17, 129].
- **FD cache with ECP-6 (FD+6)**. Frame endurance is increased allowing the failure of up to six bitcells. After the seventh failure, the frame is disabled, because an eighth

failure would no longer be recoverable [110]. This is achieved by adding six ECPs per frame to the base SECDED mechanism.

- **L2C2**. A bitcell failure is handled by disabling the corresponding byte. Cache blocks are stored compressed with BDI. It has an intra-frame wear-leveling mechanism and an LRU-Fit replacement policy; see Section 2.4.
- **L2C2+6**. An L2C2 with 6 spare bytes per cache frame; see Section 2.4.8.

Two variations of L2C2 are also tested:

- **L2C2-NWL**. An L2C2 *without* the intra-frame wear-leveling mechanism. The index generation circuit has less complexity; see Section 2.4.6. Writing always starts at the least significant live byte of the frame.
- **L2C2-BF**. It is an L2C2 with LRU-Best-Fit replacement policy instead of LRU-Fit.

Table 2.3 shows the number of storage bits per frame of the tag and data arrays, along with the percentage increments with respect to FD.

***Table** 2.3: Frame costs in bits. Percentage overhead relative to FD.*

| | SRAM Tag Array | | STT-MRAM Data Array | |
|---|---|---|---|---|
| | Bits | Overhead, % | Bits | Overhead, % |
| FD | 34 | – | 529 | – |
| FD+6 | 34 | 0 | 595 | 12.5 |
| L2C2 | 38 | 11.8 | 594 | 12.3 |
| L2C2+6 | 38 | 11.8 | 648 | 22.5 |

### 2.5.1 Experimental setup

Details of the multicore system modeled for the cycle-by-cycle simulation phase of each epoch are shown in Table 2.4. It consists of 4 cores, each with two private cache levels L1 and L2, split into instructions and data. In addition, there is a third cache level (L2C2) which is shared, non-inclusive and distributed in four banks among the cores. The coherence protocol is directory-based MOESI, and the interconnection network is a crossbar connecting the L2 private levels, the banks of the LLC and the directory. The main memory controller is located next to the directory.

We use Gem5 [83] along with the Ruby memory subsystem and Garnet interconnection network. In addition, we use NVSim for the L2C2 latency estimations [33]. The workload consists of 10 mixes randomly built by SPEC CPU 2006 and 2017 benchmarks [12, 56], leaving aside applications with very little activity on the LLC [92]. Fast-forwarding is performed for the first two billion instructions and then 200M cycles are simulated in detail. Table 2.5 shows the applications that make up each mix along with the LLC misses per kilo instructions (MPKI) of the mix, computed by dividing total cache misses by total number of

*Table 2.4: System specification.*

| | |
|---|---|
| Cores | 4, ARMv8, out-of-order (up to 8 inst/cycle), 3.5 GHz. |
| Coherence Protocol | MOESI, directory distributed among LLC banks.<br>64 B data blocks in all levels. |
| L1 | Private, 32 KB D, 32 KB I, 4 ways, LRU.<br>3-cycles load-use delay. Fetch on write miss. |
| L2 | Private, L1-inclusive, 128 KB D, 128 KB I, 16 ways, LRU.<br>11-cycles load-use delay. Fetch on write miss. |
| STT-MRAM NV-LLC | Shared, non-inclusive, 4 banks, 4MB/bank, 16 ways, LRU.<br>Load-use delay: 30-cycles frame disabling; 32-cycles L2C2.<br>Frames protected by SECDED.<br>Baseline endurance: mean $10^{11}$ wr., $cv$ = 0.2, 0.25, and 0.3. |
| Main Memory | 1 memory controller, DDR4.<br>1 channel, 8GB/channel (1200 MHz) |
| NoC | Crossbar between L2C2 banks and L2s. 32 B flits. |

*Table 2.5: Selected SPEC 2006 and SPEC 2017 applications, with suffixes 06 and 17, respectively and their MPKI. Superscript indicating top-10 memory intensive applications.*

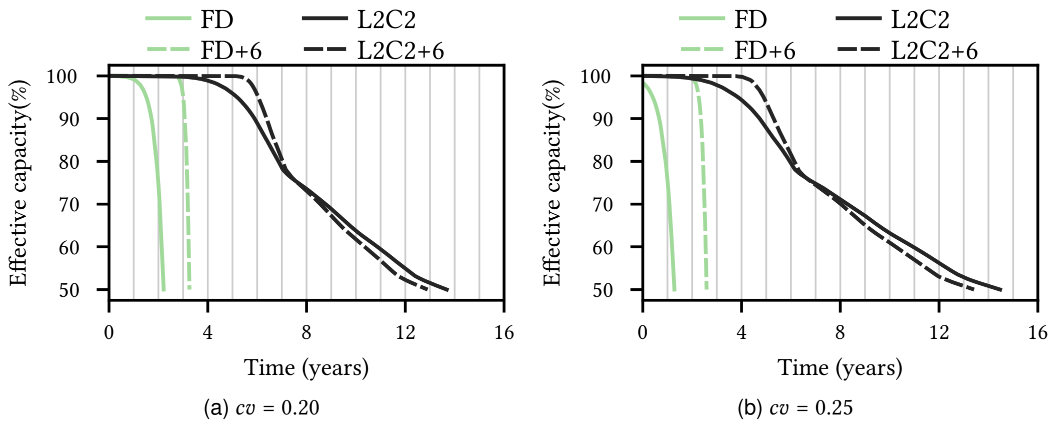
| mix | Applications | MPKI |
|---|---|---|
| #1 | zeusmp06 gobmk06 dealII06 bzip206$^{7}$ | 1.4 |
| #2 | hmmer06 bzip206$^{7}$ wrf06 roms17$^{9}$ | 2.6 |
| #3 | zeusmp06 cactuBSSN17$^{1}$ hmmer06 soplex06 | 6.1 |
| #4 | omnetpp06 astar06 milc06 libquantum06$^{4}$ | 4.9 |
| #5 | xalancbmk06$^{10}$ leslie3d06$^{3}$ bwaves17$^{6}$ mcf17$^{8}$ | 10.4 |
| #6 | lbm17$^{5}$ xz17 GemsFDTD06$^{2}$ wrf06 | 6.6 |
| #7 | cactuBSSN17$^{1}$ dealII06 libquantum06$^{4}$ xalancbmk06$^{10}$ | 7.3 |
| #8 | gobmk06 milc06 mcf17$^{8}$ lbm17$^{5}$ | 6.0 |
| #9 | xz17 astar06 bwaves17$^{6}$ soplex06 | 3.5 |
| #10 | GemsFDTD06$^{2}$ omnetpp06 roms17$^{9}$ leslie3d06$^{3}$ | 10.6 |

instructions executed by all applications in the mix. Besides, the top ten memory intensive applications, in terms of accesses per kilo instructions (APKI) are superscripted.

In order to analyze the performance evolution over time of such NV-LLCs that lose capacity due to hard faults, a forecasting procedure is employed. It allows to accurately measure the impact of different insertion policies on the evolution over time of performance, capacity of the NVM data array, and energy of the NV-LLC; taking into account the disabling of frames or bytes and the use of hardware data compression. The forecasting procedure alternates between simulation and prediction phases. The simulation phase starts reading the NV-LLC state; for instance, in L2C2 such state is the fault map of every frame, then it performs a full system simulation reporting several indexes of interest, e.g., the write rate on the frames, system instructions per cycle (IPC), and LLC hit rate. The prediction phase receives such write rates, computes the next $k$ NVM bitcells to become faulty, and update the fault map for the next simulation. In this chapter, the forecasting procedure advances in time until the NVM data array loses 50% of its capacity, but there is no problem in reaching full depletion. Chapter 4 delves into all the details of such a forecasting procedure. The IPC evolution depicted in Section 2.5.3 figures is obtained at each simulation phase, computing the arithmetic mean of the IPCs of the mixes conforming the workload. Analogously, the energy evolution depicted in Section 2.5.4 figures is obtained at each simulation phase by multiplying the energy associated to each single event by the number of events occurred in the workload.

## 2.5.2 Lifetime

Figure 2.7 shows capacity degradation, from start-up until 50% of effective capacity is lost, considering bitcells with increasing manufacturing variabilities for the four NV-LLC candidates. The effective capacity shown on the Y-axis is the one contributing to cache block storage. For example, L2C2+6 has 100% effective capacity as long as its nominal 16MB capacity is available, regardless of whether or not the spare bytes are coming into play. Besides, Table 2.6a shows $T_{50C}$, the time required to lose 50% of the nominal cache capacity.



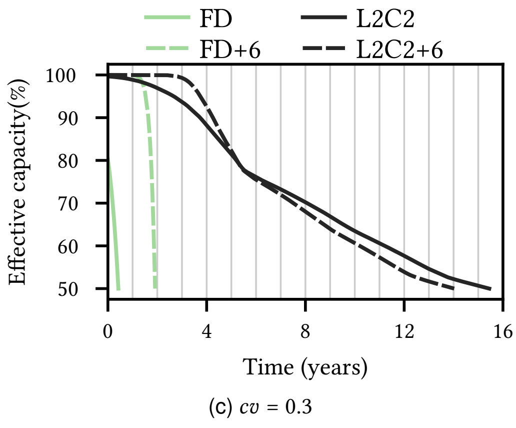


*Figure 2.7: Effective capacity evolution over time until 50% of capacity is lost.*

First of all, it can be seen that FD manufactured with high variability starts with an effective capacity that may be well below the nominal one; i.e., an FD with $cv = 0.3$ starts operating with less than 80% of nominal capacity because many frames come out of production with defective bitcells; see Figure 2.7c. FD+6, in contrast, completely solves this problem by adding redundancy. On the other hand, $T_{50C}$ decreases markedly for FD and FD+6 as the manufacturing variability increases, while for L2C2 and L2C2+6 it is the other way around; see Table 2.6a. This is due to the byte-level disabling capability of L2C2, which tolerates early byte failures and takes advantage of the later ones.

Second, compared to the sharp drop observed in frame-disabling caches, the byte-disabling ones show a much more progressive degradation of capacity, resulting in a longer $T_{50C}$. L2C2 is the longest lasting cache, in terms of $T_{50C}$ from 13.7 to 15.4 years, and FD the least, from 2.2 to 0.42 years, depending on *cv*. L2C2+6 lasts a little less than L2C2, but it is the one that maintains the nominal capacity for the longest time, namely $T_{99C}$, between 5.6 and 3.1 years, depending on *cv*; see Table 2.6b. As an example, in terms of $T_{50C}$, see Table 2.6a, L2C2 is alive 6, 11 and 37 times longer than FD for *cv* values of 0.2, 0.25 and 0.3, respectively.

Third, as time goes by, and contrary to expectations, the effective capacity of L2C2+6 is no longer greater than that of L2C2, with the curves intersecting at around 7.5 − 5.5 years, depending on *cv*. As will be seen in the next subsection the explanation is as follows: before the curves cross, the L2C2+6 system maintains a higher IPC, which implies a higher write rate and a consequent earlier degradation.

***Table** 2.6: $T_{50C}$, $T_{99C}$, and $T_{99P}$ in years; $I_{50C|5y}$ in instructions.*

| | $T_{50C}$, years. | | | $T_{99C}$, years. | | |
|---|---|---|---|---|---|---|
| *cv* | *0.2* | *0.25* | *0.3* | *0.2* | *0.25* | *0.3* |
| FD | 2.2 | 1.3 | 0.42 | 1.1 | – | – |
| FD+6 | 3.3 | 2.6 | 1.9 | 2.9 | 2.1 | 1.3 |
| L2C2 | 13.7 | 14.5 | 15.4 | 3.9 | 2.4 | 0.9 |
| L2C2+6 | 12.9 | 13.4 | 14.0 | 5.6 | 4.4 | 3.1 |

(a) (b)

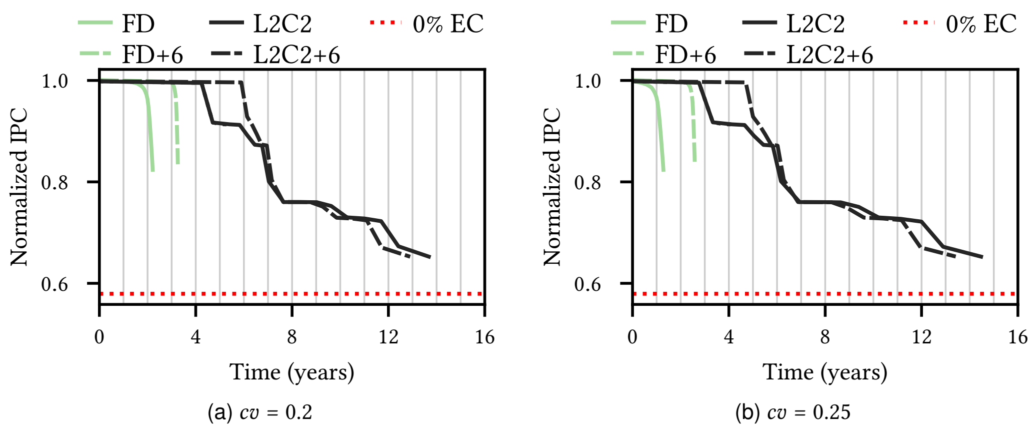
| | $T_{99P}$, years. | | | $I_{50C|5y}$, instr. $\times 10^{18}$ | | |
|---|---|---|---|---|---|---|
| *cv* | *0.2* | *0.25* | *0.3* | *0.2* | *0.25* | *0.3* |
| FD | 1.7 | 0.65 | – | 0.89 | 0.51 | 0.16 |
| FD+6 | 3.1 | 2.4 | 1.6 | 1.32 | 1.05 | 0.77 |
| L2C2 | 4.3 | 2.8 | 0.82 | 2.01 | 1.96 | 1.90 |
| L2C2+6 | 5.9 | 4.7 | 3.5 | 2.03 | 2.03 | 1.98 |

(c) (d)

### 2.5.3 Performance

Figure 2.8 shows the IPC evolution over time from start-up until 50% of effective capacity is lost for the NV-LLC candidates. The IPC is normalized to the IPC of a system with an NV-LLC with all bitcells operational. The bottom dotted red line (0% EC) represents the IPC of a system with a fully impaired NV-LLC, i.e. with zero effective capacity.

First, after losing 50% of capacity, the IPC in FD caches is around 20% higher than that in L2C2 caches. This is because having 50% effective capacity with FD or FD+6 implies that 50% of frames can store any block, whereas with L2C2 and L2C2+6, it implies that the



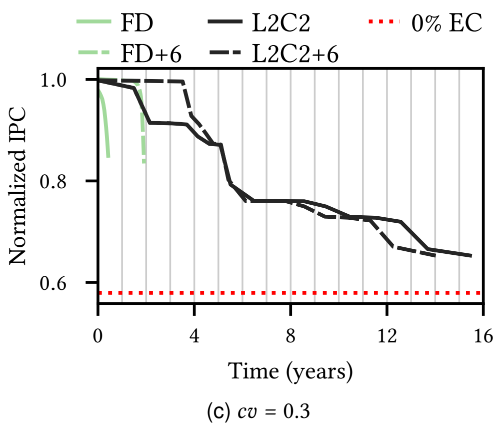


***Figure** 2.8: Normalized IPC evolution over time until 50% of capacity is lost.*

capacity of all frames has been reduced and therefore some blocks cannot be stored in any frame.

Second, consistent with the effective capacity degradation, the IPC drops later and more gradually in L2C2 and L2C2+6. The steps seen in their lines correspond to periods in which the possibility of storing blocks of a given compression encoding has been lost.

Third, the crossings in the IPC and capacity curves occur at the same times. After these crossings, L2C2 performs slightly better and lasts slightly longer than L2C2+6. The reason is to be found in the first 4 − 6 years of operation of L2C2+6 at maximum performance, years that, compared to L2C2, cause a higher write wear.

And fourth, in the first years of operation L2C2+6 keeps the maximum performance, L2C2 loses it progressively, and FD and FD+6 loses it abruptly. The index $T_{99P}$, the time during which performance holds above 99% of the maximum allows to quantify these facts; see Table 2.6c. L2C2+6 excels at $T_{99P}$ for all $cv$ values, with L2C2 in second place, except for $cv = 0.3$, where FD+6 is better.

From the above analysis, L2C2+6 seems to be the best candidate, followed by L2C2, and at some distance FD+6.

To get more insight, we propose to measure the work performed by the different organizations using the aggregate number of instructions executed by the four cores, with a utilization of 100%, until a certain wear-out condition is reached. We compute this value by integrating the IPC curve. According to Belkhir et al., the average lifetime of a server is three to five years [8], we thereby propose the index $I_{50C|5y}$ which measures the number of instructions executed until 50% of the capacity is exhausted or until five years have elapsed, whichever is earlier; see Table 2.6d.

Regarding this index, we can say that the increase in manufacturing variability is very bad for frame disabling, with reductions of 82 and 42% of $I_{50C|5y}$ in FD and FD+6, going from *cv* 0.2 to 0.3. In contrast, that same increase in *cv* slightly reduces $I_{50C|5y}$ in L2C2 and L2C2+6 by 5.5 and 2.5%, respectively.

In short, L2C2+6 offers the best performance in all indexes, with an additional storage cost over L2C2 and FD+6 of less than 10%. The second option, cheaper but with less performance, is L2C2, which requires about 12.3% more data array storage than FD, the base option without redundancy.

### 2.5.4 Energy

Figure 2.9 shows the energy evolution over time of the different NV-LLCs from start-up until 50% of effective capacity is depleted. The energy is normalized to the energy of a FD cache with all bitcells operational. The energy overheads of L2C2 caches regarding the metadata and the rearrangement circuits introduced in Section 2.4.7 and Table 2.2 are taken into account for the analysis.

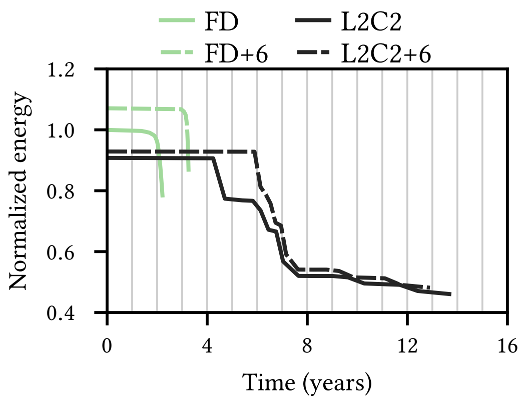
First, the energy consumed by L2C2 is 9% less than that of FD at the beginning of the cache life, see Figure 2.9a. Although the metadata overhead increases the static power and the rearranging circuits increase both the dynamic read and write energy, the selective writing thanks to the data compression and the write mask offsets these increases by significantly reducing the dynamic write energy. This allows us to conclude that, despite the hardware overheads, our design does not incur in a higher energy consumption.

Second, the energy drops are consistently aligned with the performance drops observed in Figure 2.8 for all the considered configurations. The capacity degradation leads to a loss of performance, which, in turn, reduces the activity of the NV-LLC.

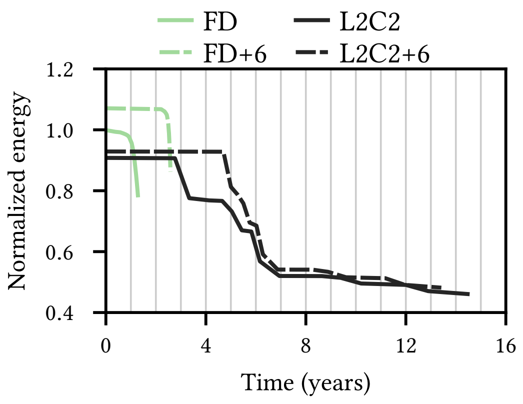
Besides, both FD+6 and L2C2+6 increase the energy consumption compared to FD and L2C2, respectively. This is because the redundant information increase the static power of the storage structures. Therefore, the prolonged higher performance of the redundant versions is achieved at the cost of a higher energy consumption.

### 2.5.5 Intra-frame wear-leveling

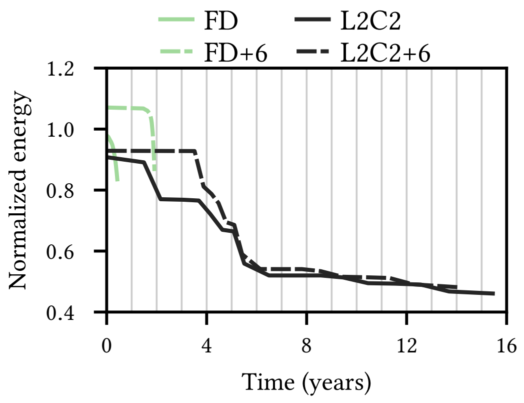
In this experiment we aim to see the importance of the intra-frame wear-leveling mechanism. In an L2C2 without intra-frame wear-leveling, there is an imbalance between the number of writes that receive the low order bytes and the high order bytes of a frame. Concretely, higher order bytes will not be written if the block compressed to some extent. This imbalance will make lower order bytes receive more write operations than higher order ones so they will become faulty before than in a cache with intra-frame wear-leveling.



(a) $cv = 0.2$



(b) $cv = 0.25$



(c) $cv = 0.3$

***Figure** 2.9: Normalized energy evolution over time until 50% of capacity is lost.*

Figure 2.10 shows the IPC evolution until the NV-LLC loses 50% of its effective capacity for $cv = 0.2$ of L2C2 and L2C2-NWL, an L2C2 whithout the intra-frame wear-leveling mechanism. The IPC of L2C2-NWL starts dropping at 3.7 years while L2C2 IPC drops at 4.3 years (16% later). This temporal shift linking points of equal performance is evident throughout the duration studied, being around one year on many occasions.

### 2.5.6 Fit vs. Best-Fit replacement

In L2C2 an alternative replacement policy to LRU-Fit is LRU-Best-Fit, which consists of choosing the smallest LRU frame capable of holding the incoming compressed block; see L2C2-BF in Figure 2.11. In principle, LRU-Best-Fit could be advantageous since it would preserve frames with larger capacity from writes, allowing in the long term the hosting of blocks with low compression capacity; see Section 2.4.1. However, L2C2-BF takes 8.9 years to lose 50% of its capacity, while L2C2 reaches the same loss at 13.7 years, i.e. 54% longer. Besides, the IPC drop L2C2-BF experiences at the early stages (0 – 2 years) is even more pronounced than that of FD. The explanation for both effects is that when the first frame in a set experiences the first byte failure, all the compressible blocks addressed to this set,

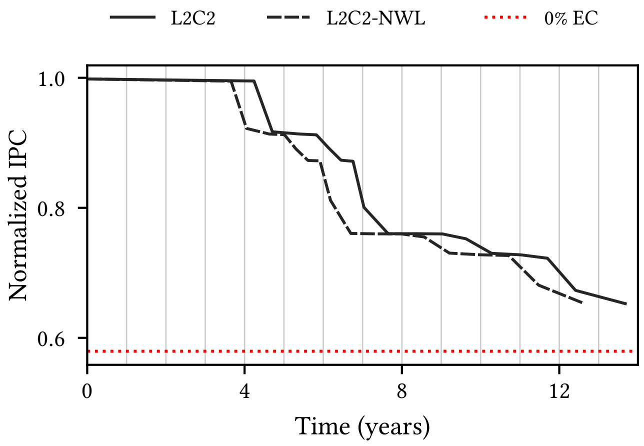


***Figure** 2.10: IPC evolution until losing 50% of capacity of an L2C2 without intra-frame wear-leveling mechanism, L2C2-NWL, for cv = 0.2.*

78% of the total, will be allocated to this recently degraded frame; see Figure 2.2 in page 6. This incurring in substantial conflict misses that degrade performance.

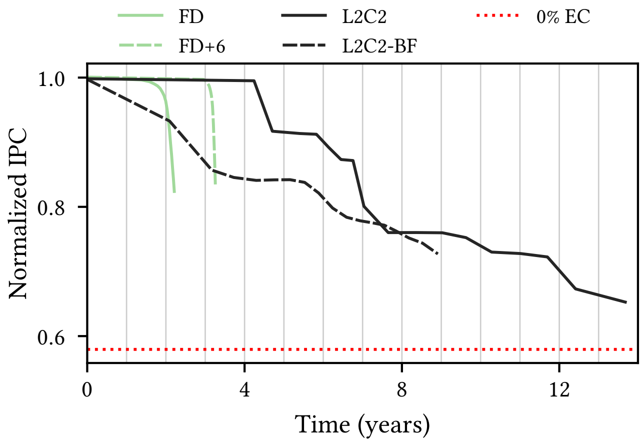
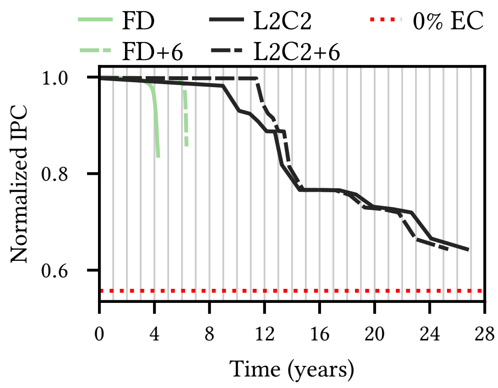


***Figure** 2.11: IPC evolution until losing 50% of capacity of an L2C2 with LRU-Best-Fit replacement policy, L2C2-BF, for cv = 0.2.*

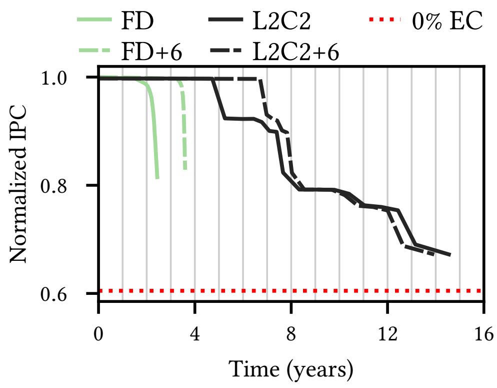
### 2.5.7 Sensitivity analysis

To further add generality to the results presented so far, we elaborate on three aspects; see Figure 2.12. First, the LLC bank size is increased from 4 to 8 MB per bank. Second, the system is scaled by a factor of 2, going from 4 to 8 cores, from 4 to 8 banks of NV-LLC and from 1 to 2 main memory controllers. And third, the workload mixes are changed, including only the top ten memory intensive applications; see applications with superscript in the Table 2.5.

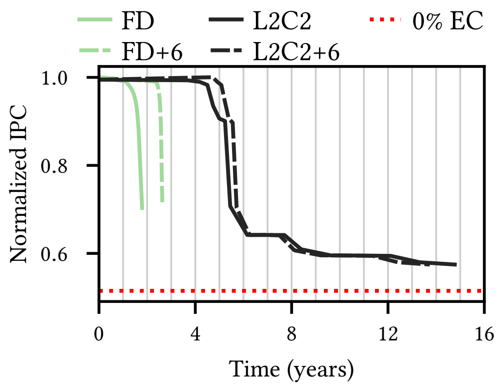
Doubling cache capacity with the same number of cores extends performance over time to a similar amount across all cache organizations. For example, for L2C2+6, $T_{50C}$ goes from 12.9 to 25.3 years when increasing size from 16 to 32 MB; see Figure 2.8a vs. Figure 2.12a.



(a) 32MB caches.



(b) 8 cores.



(c) Memory-intensive.

***Figure** 2.12: IPC evolution until losing 50% of capacity of FD and L2C2 for cv = 0.2, doubling cache size (a), doubling the number of cores while keeping the same 4MB/core (b), and considering the most memory-intensive applications (c).*

By scaling the system, simultaneously doubling number of cores, cache size and memory bandwidth, the performance-time curves for all cache organizations maintain their shape; see Figure 2.8a vs. Figure 2.12b. This is an expected conclusion, which reinforces the possibility of incorporating L2C2-type caches in future generations of on-chip multiprocessors.

When considering more memory intensive applications, a first observation is that the performance at full capacity exhaustion is lower, which indicates, not surprisingly, a higher dependence of performance on the quality of the memory hierarchy; see the red baselines (0% EC) in Figure 2.8a vs Figure 2.12c. In addition, the performance drop is sharper and occurs earlier. For example, for L2C2+6 the first drop is one year earlier and the relative IPC drops from 0.76 to 0.64. Again, it can be reasoned that applications that exhibit intensive LLC usage are more sensitive to capacity loss, so overall system performance is more affected.

In summary, this sensitivity analysis shows that the results are consistent when varying two significant dimensions, capacity and workload.

## 2.6 Concluding remarks

In this chapter, L2C2 has been introduced. It is a new fault-tolerant NV-LLC organization that achieves per-byte write rate reduction without performance loss and allows compressed blocks to be placed in degraded frames. L2C2 evenly distributes the write wear within each frame, uses an appropriate replacement policy, and inherently allows adding redundant capacity in each cache frame, further extending the time in which the cache remains without performance degradation. Data compression and decompression circuits have been synthesized, considering intra-frame wear-leveling, concluding that their inclusion seems very feasible in terms of area, power and latency.

Our evaluation shows that, with an affordable hardware overhead, L2C2 achieves a large lifetime improvement compared to a reference NV-LLC provided with frame disabling. The lifetime is multiplied by a factor from 6 to 37 times depending on the variability in the manufacturing process. Increasing redundancy significantly increases the time to loss of performance by one to two years in all configurations, regardless of the variability in the manufacturing process. However, it does not increase the lifetime of the L2C2.

*Money ain't got no owners. Only spenders.*

Omar Little, The Wire.

# 3 Compression-aware and performance-efficient insertion policies for long-lasting hybrid LLCs

*As seen in the previous chapter, NVMs can potentially replace large SRAM memories such as the LLC. However, despite recent advances, NVMs suffer from higher write latency and limited write endurance. NVM-SRAM hybrid LLCs are proposed to combine the best of both worlds. Several policies have been proposed to improve the performance and lifetime of hybrid LLCs by intelligently steering the incoming LLC blocks into either the SRAM or NVM part, regarding the cache behavior of the LLC blocks and the SRAM/NVM device properties. However, these policies neither consider compressing the contents of the cache block nor using partially worn-out NVM cache blocks.*

*This chapter proposes new insertion policies for a hybrid-extended version of L2C2, which is a byte-level fault-tolerant hybrid LLC that collaboratively optimize for lifetime and performance. Specifically, we leverage data compression to utilize partially defective NVM cache entries, thereby improving the LLC hit rate. The key to our approach is to guide the insertion policy by both the reuse properties of the block and the size resulting from its compression. A block is inserted in NVM only if it is a read-reuse block or its compressed size is lower than a threshold. It will be inserted in SRAM if the block is a write-reuse or its compressed size is greater than the threshold. We use set-dueling to tune the compression threshold at runtime. This compression threshold provides a knob to control the NVM write rate and, together with a rule-based mechanism, allows balancing performance and lifetime.*

*Overall, our evaluation shows that, with affordable hardware overheads, the proposed schemes can nearly reach the performance of an SRAM cache with the same associativity while improving lifetime by 17× compared to a hybrid NVM-unaware LLC. Our proposed scheme outperforms the state-of-the-art insertion policies by 9% while achieving a comparative lifetime. The rule-based mechanism shows that by compromising, for instance, 1.1% and 1.9% performance, the NVM lifetime can be further increased by 28% and 44%, respectively.*

## 3.1 Introduction

The ever-growing working set sizes of emerging application domains such as machine learning and artificial intelligence require larger on-chip LLCs. Increasing the LLC capacity is also imperative as the number of cores sharing it grows, because it is the last line of defense of the processor against costly off-chip memory accesses. However, with the deceleration of Moore's law, the increase in the LLC capacity has stagnated [57]. The scaling of conventional SRAM-based LLCs significantly increases the leakage power consumption and is becoming prohibitive in terms of both capacity and area [74]. Therefore, recent research advocates employing emerging NVM technologies to increase the LLC capacity.

Emerging NVM technologies such as STT- and SOT- MRAM, PCM, ReRAM, and RTM have shown great promise to replace or augment conventional SRAM and DRAMs technologies. In the last decade, some NVM technologies have matured greatly and have made their way into the memory hierarchy [62, 115]. Compared to conventional SRAM technologies, NVMs, particularly MRAMs are attractive alternatives for large size LLCs because they are extremely energy efficient, offer larger densities, and SRAM-competitive read latencies [5, 19, 71, 74]. However, without proper buffering the slow write operation on NVMs can degrade performance by throttling subsequent critical reads, potentially leading to core stalls. In addition to the read/write asymmetry, most NVMs also have a limited endurance, i.e., the number of writes that each bitcell supports, until it deteriorates and loses its retention capacity is limited and can be approximated by a normal distribution with a mean that can vary between $10^6$ and $10^{12}$ [24, 44, 121, 123, 124]. Many device, circuit, and architectural optimizations have been proposed to mitigate the impact of the write operations on the NVM-LLC performance and lifetime [52, 74, 94]. However, these solutions increase the overall power consumption and reduce the NVM capacity, thereby offsetting the NVMs benefits.

Recent proposals combine the best of both worlds, i.e., performance and endurance of SRAM/DRAMs with the energy efficiency and density of STT-MRAM to implement *hybrid* LLCs [5, 49, 71, 84]. MRAMs, compared to other NVM technologies, offer better endurance and SRAM comparable read latencies with higher density. However, it still suffers from higher write latencies and limited endurance compared to conventional SRAM/DRAMs technologies. Therefore, the performance, energy, and lifetime improvements of these hybrid proposals are associated with the reduction in number of write requests to the NVM. Thus, various techniques have been proposed to identify and steer write-intensive blocks towards the SRAM part and read-intensive blocks towards the NVM part [19, 84]. The identification of read- and write-intensive blocks is either performed with address-based predictors that sample LLC accesses [5] or with predictors using counters and threshold values for LLC block accesses [73].

The asymmetric read-write operations in NVMs have also motivated novel insertion policies. For instance, Luo et al. propose Thrashing Aware Placement (TAP) which classifies LLC write requests into demand-writes, prefetch-writes, and clean/dirty thrashing-writes [84]. Thrashing requests are routed to the SRAM part to reduce the LLC energy consumption and improve lifetime. Compared to the LRU replacement policy, their proposal reduces the energy consumption by 25%. Similarly, Cheng et al. propose LHybrid [19], a loop-block aware policy to insert only clean blocks that are frequently reused (loop-blocks) in the NVM part, protecting them from non-loop-blocks when a victim is selected for replacement. LHybrid significantly reduces write traffic and improves LLC lifetime. However, in these previous proposals, the LLC lifetime improvement is only achieved by conservative insertion in the NVM part, which limits LLC performance.

In addition to specific insertion and replacement policies, a different class of optimization techniques improves the NVM lifetime and performance by decreasing the average number of bits written in each write request [25, 95, 96]. In particular, compression can increase effective main memory capacity and reduce bandwidth utilization by $2 - 4\times$ [1]. Unfortunately, compression has received only little attention in the context of hybrid LLCs. In particular, the behavior of the state-of-the-art insertion policies for hybrid-LLCs enhanced with compression are yet to be investigated.

Figure 3.1 shows a forecast of the performance evolution of a hybrid LLC over time, until the capacity of the NVM drops to 50%. Twelve and four ways have been devoted to NVM and SRAM storage, respectively. Further details on the methodology and workload are discussed in Section 3.5.1. The baseline hybrid (BH) LLC configuration manages a single LRU list for all ways in a cache set. The insertion policy does not distinguish between NVM and SRAM parts and incoming blocks are written to the LRU way, regardless of its technology. BH initial performance is excellent, but the write wear on the NVM part leads to 50% of its capacity being exhausted in less than three months.

Compared to BH, LHybrid [19], thanks to its selective insertion policy, improves LLC lifetime by more than $19\times$, but at the cost of significant performance degradation ($> 11\%$). TAP [84] sacrifices even more performance in exchange for a lifetime improvement of $39\times$.

This chapter bridges the performance and lifetime disparities between BH and LHybrid approaches by proposing *compression with Set Dueling* (CP_SD), a hybrid insertion policy that combines data compression and block reuse information. Besides, the NVM part tolerates byte-level faults and is provided with a block rearrangement circuitry. As can be seen in Figure 3.1, CP_SD maintains for almost two years 97% BH performance, reaching 50% capacity exhaustion in about three years and nine months. Our solution strikes a good balance in the performance vs. lifetime tradeoff, prioritizing performance without neglecting lifetime. Moreover, this chapter also proposes a rule-based mechanism to further tune this tradeoff: CP_SD_Th4 and CP_SD_Th8 in Figure 3.1 trade 1.1% and 1.9% performance in exchange for 28% and 44% NVM lifetime improvement, respectively.

Specifically, this chapter makes the following contributions to shared hybrid NVM-SRAM LLCs whose NVM part tolerates byte-level faults and leverages data compression:

- A novel insertion scheme that places cache blocks into either the SRAM or NVM part of the LLC, considering the read-reuse, write-reuse and compression features of cache blocks. In the NVM part, the replacement algorithm considers an NVM

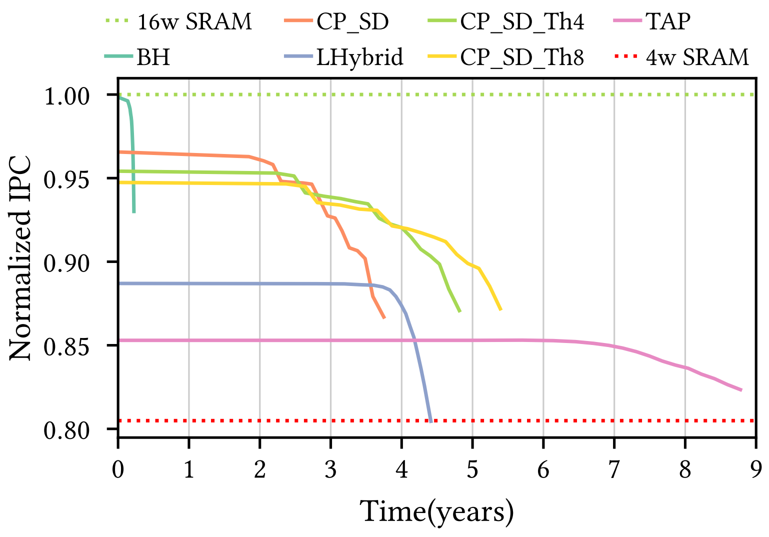


***Figure** 3.1: Performance vs. time for various hybrid LLCs until the NVM part loses 50% capacity. The write endurance of NVM bitcells follows a normal distribution of $\mu = 10^{10}$ and $cv = 0.2$. Bounds of SRAM-only LLCs are also plotted.*

fault-map and the compressed size of the incoming LLC block, replacing the LRU block from the frames the incoming block can fit in.

- A threshold-based mechanism tunes the write-traffic to the NVM part, thereby allowing to explore the tradeoff between performance and lifetime. We propose to use Set Dueling [102] to capture the runtime behaviour of the workload and allow more (or less) compressed blocks to be inserted in the NVM part. The sample cache sets collect the number of writes and the number of hits. Based on these counters, a rule-based decision mechanism balances lifetime and performance.

- The forecasting procedure, which is introduced in Chapter 4, is adapted to the hybrid LLC scenario. This procedure tracks NVM aging, providing the temporal evolution of performance and capacity. It allows to analyze all dimensions of the hybrid LLC design.

- For a fair comparison, it is necessary to test existing insertion policies on NVM caches that lose capacity due to aging. Therefore, the state-of-the-art LHybrid and TAP policies [19, 84] are implemented in a fault-aware environment, extended with *frame-disabling* to tolerate hard-errors [17, 129].

- For evaluation, we consider multi-programmed workloads of memory-intensive applications from the SPEC 2006 and SPEC 2017 suites. The evaluation environment combines three elements: a fast architectural simulator, developed in Chapter 5, a detailed cycle-level simulator [83] and the forecasting procedure mentioned above. We present a comprehensive analysis and evaluation of our novel schemes and their comparison to the state-of-the-art. We show that our proposals consistently and significantly outperform the state-of-the-art in all performance metrics.

The rest of the chapter is organized as follows. Section 3.2 motivates this study. Section 3.3 describes the microarchitecture of the hybrid LLC. Section 3.4 describes the proposed insertion policies together with the Set Dueling mechanism. Section 3.5 presents the results of our insertion policies against the state-of-the-art. Finally, Section 3.6 concludes the chapter.

## 3.2 Related work and motivation

### 3.2.1 State-of-the-art hybrid LLC insertion policies

This section outlines LHybrid [19] and TAP [84], two state-of-the-art insertion policies for hybrid LLCs.

LHybrid [19] classifies LLC blocks into *loop-blocks* and *non-loop-blocks*. Cache blocks that are not modified during their round trips between L1/L2 and LLC, i.e., read-only blocks that show reuse in the LLC are referred to as loop-blocks (LBs). They are of utmost importance in the context of hybrid LLCs because they are ideal residents of the NVM part. LHybrid strives to keep as many LBs in the NVM part as possible and to steer non-loop-blocks (NLBs) into the SRAM part.

The LHybrid insertion scheme works as follows. All blocks in LLC and L2 are tagged as LB or NLB and this tag is supplied along with the block in both directions. Initially, all blocks entering L2 from the main memory are marked as NLB. A block evicted from L2, marked as NLB and not present in the LLC, is inserted into the SRAM part. Conversely, a block evicted from L2 and tagged as LB, if not in the LLC, is inserted into the NVM part. A read request from L2 that hits LLC implies a previous eviction from L2, and thus a reuse. This read request will tag the block as LB if, and only if, the block is clean. In the LHybrid replacement scheme, for the NVM part, the LRU block is simply evicted (local replacement). In SRAM, the replacement policy first searches for LBs, and if found, the most recent LB, in LRU order, is migrated to the NVM part; otherwise, the LRU block is evicted.

Similar to LHybrid, TAP [84] defines *thrashing-blocks* as blocks that have hit in the LLC more than $TH_{thrash}$ times. TAP only inserts clean thrashing-blocks to the NVM part because they are expected to stay longer in the LLC, preventing energy-hungry NVM write operations from other blocks. In terms of NVM insertions, TAP is more conservative than LHybrid, see Figure 3.1, because a block needs to show reuse more than once (unlike the LHybrid LB) to be inserted in the NVM part. We thus use LHybrid as the state-of-the-art reference policy so that results are more comparable in terms of performance.

### 3.2.2 Motivation: quantitative analysis of hybrid LLC insertion policies

As described in the previous section, state-of-the-art insertion policies conservatively target the NVM part, which extends lifetime but sacrifices performance. To demonstrate this, we evaluate ten multi-programmed workloads from the SPEC 2006 and SPEC 2017 benchmarks on a hybrid LLC having 12 NVM-ways and 4 SRAM-ways (see Section 3.5.1 and Table 3.2 for more details) and show the impact of different configurations on the LLC performance and lifetime in Figure 3.1. For comparison, we use SRAM-only LLC configurations with 16-ways (best-case) and 4-ways (worst-case, as if the 12-NVM ways were faulty), to determine the

upper and lower bounds on the hybrid LLC performance. Both configurations employ the LRU replacement scheme and are compared to the following.
**BH**, that is NVM-unaware and naively fills data into NVM and SRAM ways implementing a global LRU replacement policy and frame-disabling, achieves performance similar to that of a 16-way SRAM cache, the expected upper limit. The small performance loss compared to an SRAM cache is exclusively due to the increased latency in the STT-MRAM ways, since the contents of both caches are exactly the same. However, for a mean endurance of $10^{10}$, it takes less than three months for the NVM part to lose 50% of its effective capacity.
**LHybrid**, which conservatively inserts into the NVM by steering only the loop-blocks into it and employs a local loop-block-aware replacement scheme. As a result, it improves the NVM lifetime by 19.7× compared to BH but at the cost of a significant 11% performance decrease.
**TAP** is more conservative than LHybrid. Blocks must show higher level of reuse (clean thrashing-blocks) to be inserted in the NVM. It improves the hybrid cache lifetime by 39× compared to BH in exchange for 15% performance drop.

To reduce these wide disparities between performance and lifetime of different configurations, this chapter investigates hybrid LLC designs that achieve near-BH performance and near-LHybrid lifetime by jointly optimizing for both metrics.

## 3.3 Fault-tolerant hybrid LLC architecture

Hybrid LLC designs require intelligent insertion policies supported by the underlying microarchitecture. L2C2, which is a monolithic NV-LLC, see chapter 2, laid the groundwork for our hybrid LLC design. This section presents our microarchitectural extensions to L2C2 in order to get the most out of the insertion policies.

Figure 3.2 shows a high-level overview of the proposed hybrid LLC design. Our hybrid LLC, as in L2C2 -see 2.4.1- employs data compression together with byte-disabling and the intra-frame wear-leveling mechanism to mitigate the effective capacity drop due to NVM hard-faults. It also features a non-inclusive hierarchy and a more sophisticated insertion mechanism that will be introduced in the following section. It is also assumed that all memory structures are provided with ECCs. Note that this ECC protection does not bring any additional overhead as they are needed by state-of-the-art designs to deal with faults and are already present in SRAM LLCs to cope with soft and transient faults [3, 9, 44, 66, 116, 132]. The main difference regarding L2C2 is that the data array, see Figure 3.3, is split in NVM ways and SRAM ways, typically, with a factor of three NVM ways for every SRAM way [19, 84].

Filling a block in the proposed hybrid LLC follows a flow similar to that of L2C2, which is comprehensively described in Section 2.4.4. When a block arrives to the hybrid LLC, the block is first compressed. Then, the compressed block (CB) is protected with ECCs and, in parallel to this, the insertion engine selects the victim frame. In case that the block must be written to the NVM part, the replacement algorithm will look for the target frame among those with an effective capacity greater than or equal to the incoming compressed block (LRU-Fit) [45]. Besides, the compressed block is properly rearranged regarding the intra-frame wear-leveling and the fault map. On the contrary, if the block is directed to

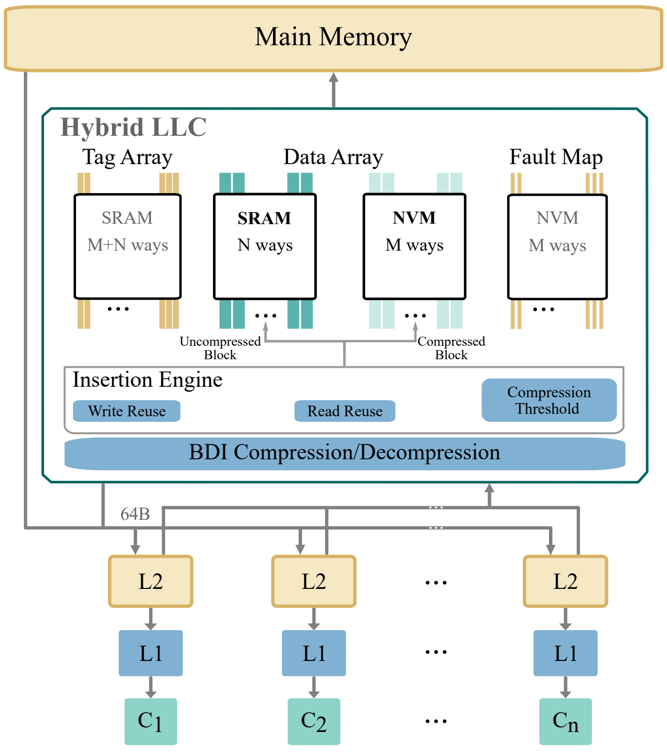


***Figure** 3.2: High-level overview of the hybrid LLC organization.*

the SRAM part, a conventional LRU is employed for replacement, and the block is stored uncompressed.

**Latency overheads.** Using VLSI synthesis for 16 nm, the block rearrangement circuitry has proven to be feasible in terms of latency (incurring 0.33 and 0.38 ns for writing and reading, respectively) as well as area and power consumption, see Section 2.4.7. The BDI decompressor, that is a single instruction multiple data (SIMD)-style vector adder [75, 99], incurs a 2-cycle latency overhead. Note that all the competing mechanisms need SECDED for hard-fault detection. For this reason, the latency incurred by SECDED is not accounted for in the overheads.

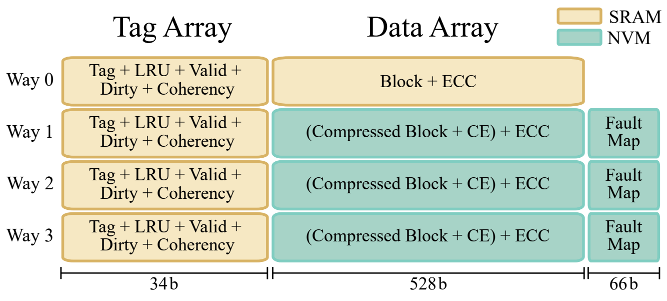


***Figure** 3.3: Example of a four-way cache set split into three NVM ways and one SRAM way, showing fields and their sizes.*

## 3.4 Compression-aware insertion policies

Data compression reduces the size of the incoming blocks to the LLC and thereby reduces number of bytes written to cache frames. Together with byte-disabling, compression can be used to allocate reduced size blocks to partially defective NVM frames, always taking care to level the write wear among the remaining non-faulty bitcells. This section describes how block features such as compressed size, read-reuse and write-reuse information can be leveraged to develop performance-efficient and lifetime-aware content management policies.

### 3.4.1 Naive compression-aware (CA) insertion

We denote blocks whose compressed size is lower than or equal to a given compression threshold ($CP_{th}$) as *small blocks* and blocks that are either incompressible or whose compressed size is greater than the threshold as *big blocks*. Intuitively, the greater the compression ratio a block achieves, the less harmful it is to write it on the NVM part. A naive *compression-aware* (CA) insertion policy, therefore, directs small blocks to NVM ways and big blocks to SRAM ways. Both NVM and SRAM ways follow a local LRU replacement policy.

CA may lead to performance degradation. The performance loss occurs when there is an imbalance between the number of references to blocks allocated in NVM and SRAM ways.

For instance, 100% of the cache blocks are incompressible in the benchmarks `xz17` and `milc`, see Figure 2.2, and as a result CA will direct all blocks to the SRAM ways. On the contrary, in benchmarks such as `GemsFDTD` and `zeusmp`, where almost all cache blocks are highly compressible (HCR), the CA policy will only insert blocks into NVM ways. In both cases, one part of the LLC is over-referenced, experiencing many misses and leading to performance degradation.

**CA evaluation.** The light green bars (CA) in Figure 3.4 show the LLC hit rate for different $CP_{th}$ values, normalized to BH hit rate. Unless otherwise mentioned, all results in this and the following sections are averaged across ten multiprogrammed mixes of four randomly selected applications from the employed subset of SPEC 2006 and SPEC 2017, see Table 3.4. The normalized hit rate varies between 0.89 and 0.99, with the highest figure being obtained for a $CP_{th}$ of 58. With this $CP_{th}$ value, only uncompressed blocks are inserted into SRAM and the rest into NVM. The attained hit rate is very close to BH, which indicates that this $CP_{th}$ value achieves a block distribution with only little conflict misses compared to BH and, therefore, is well balanced.

The light green bars (CA) in Figure 3.5 show the number of bytes written (BW) to the NVM part for different $CP_{th}$ values normalized to the values obtained in BH. As can be seen, the $CP_{th}$ has a considerable impact on the number of BW in the NVM part, varying between 5% and 80% of the writes for BH. With a $CP_{th}$ of 58, the best in terms of hit rate from Figure 3.4, the number of BW is still 40% lower than the BH cache.

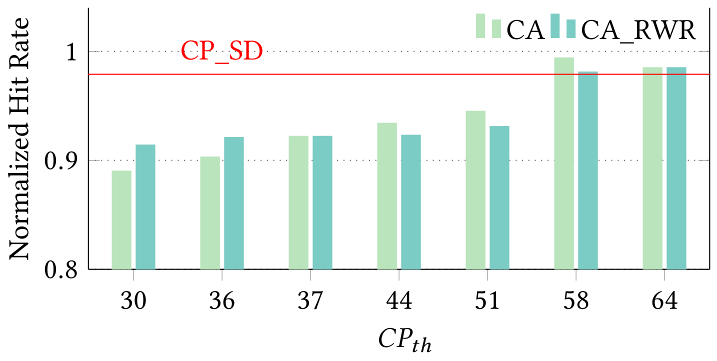


***Figure** 3.4: Hybrid LLC hit rate with different $CP_{th}$ normalized to BH.*

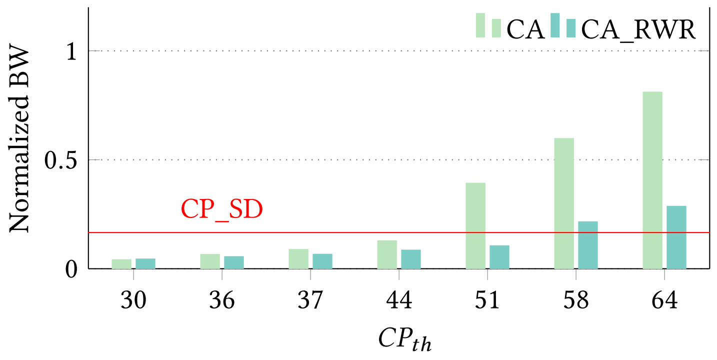


***Figure** 3.5: Normalized BW: average number of bytes written per frame in the NVM part, normalized to BH, varying $CP_{th}$.*

### 3.4.2 Read and write reuse aware insertion

As discussed in Section 3.2, several works have shown that content management based on block reuse properties can reduce the number of writes in NVM caches [19, 84]. We now discuss how to incorporate read and write reuse along with the CA insertion policy: CA + read- write- reuse (CA_RWR).

We classify blocks into three categories based on their reuse properties: blocks that have not yet demonstrated any reuse, blocks with read reuse, and blocks with write reuse. Initially, all blocks are classified as non-reused when copied from the main memory to the cache hierarchy. A hit in the LLC classifies a block into either a read-reused block if it has not been modified or a write-reused block if it has been written at least once. Notice that our read-reuse class corresponds to the loop-blocks in LHybrid while write-reuse and non-reuse blocks correspond to non-loop-blocks.

Table 3.1 summarizes CA_RWR, our proposed insertion policy that places blocks in either SRAM or NVM depending on the size of the compressed block and its reuse type:

- Blocks that show *read reuse* are candidates to stay longer in the LLC. Inserting them in the NVM part is beneficial, regardless of the compressed size, because they will stay longer in the LLC, preventing other writes in the frame.

- Blocks that show *write reuse* are candidates to stay for a short time in the LLC due to the invalidate-on-hit coherence policy of LLC requests with write permission (GetX), see Section 2.4.1. Such dirty blocks will be inserted back into LLC when they are evicted from L2. Therefore, they should be placed in SRAM since they are candidates for multiple LLC writes.

- Blocks *without reuse* are inserted either into SRAM or NVM, depending on their compressed size, i.e., small blocks are inserted in NVM and big blocks in SRAM.

Note that NVM frames render partially defective due to write operations. Therefore, a block directed to NVM that does not fit in any NVM frame, because its compressed size is bigger than any of their effective capacities, will be placed in SRAM.

***Table*** *3.1: CA_RWR insertion policy*

| | | Compressed size | |
|---|---|---|---|
| | | Small | Big |
| Reuse | no | NVM | SRAM |
| | R | NVM | NVM |
| | W | SRAM | SRAM |

As mentioned above, blocks are initially inserted into SRAM or NVM depending only on their compressed size, since they have not yet shown any reuse. In fact, many of them will be evicted from the LLC without being reused. But for those that do show reuse, the final destination will depend on the type of reuse, read or write. Therefore, it is sometimes necessary to migrate blocks between SRAM and NVM arrays: i) blocks initially stored in NVM that show reuse on write, and ii) blocks initially stored in SRAM that show reuse on read. On the one hand, reuse on write is detected when a GetX request hits in LLC and the block is invalidated. Later, when the block is evicted from L2, it will be inserted into SRAM as a write-reused block (regardless of its compressed size). On the other hand, a block initially stored in SRAM that is reused on read remains in SRAM until it is evicted (due to a replacement). At that time, the block is migrated to NVM.

**CA_RWR evaluation.** The dark green bars (CA_RWR) in Figures 3.4 and 3.5 show the normalized LLC hit rate and BW on a CA_RWR cache, varying $CP_{th}$. Compared to the cache with CA policy, the reuse information in the block insertion policy has a relatively small impact on the hit rate but a noticeable impact on BW, especially for high $CP_{th}$ values. The hit rate is better than the CA cache for small values of $CP_{th}$ and marginally worse for high values of $CP_{th}$; specifically, the CA_RWR hit rate only increases by 1.9% for $CP_{th}$ values between 30 and 51 and increasing to 5.4% for $CP_{th}$ 58. Compared to the CA, the relative decrease in BW for CA_RWR is significant, reaching 73% for $CP_{th}$ 51. Even though BW varies between 4.4% and 28.6% as $CP_{th}$ varies between 30 and 64.

### 3.4.3 CP_SD insertion: Set Dueling for performance

Figures 3.4 and 3.5 illustrate in simplified form the influence of $CP_{th}$ on the normalized hit rates and BW, averaging the results for the entire workload and assuming full capacity in the NVM part. However, for the best $CP_{th}$ selection, one must delve deeper into the impact of two key factors. First, applications may exhibit different behaviors throughout their execution, and second, as the NVM part ages, its capacity decreases.

To analyze the impact of workload time variability and NVM capacity loss we will divide the workload execution time into epochs of fixed duration and calculate the hit rate achieved in each epoch with each $CP_{th}$ value. The different bar colors in Figure 3.6 show the percentage of epochs for which each $CP_{th}$ value achieves the highest number of cache hits. Specifically, Figure 3.6a presents the distributions of optimal $CP_{th}$ varying the NVM part capacity from 100 to 50%, and Figure 3.6b presents the distributions of optimal $CP_{th}$ for each workload in an LLC with 100% capacity in the NVM part.

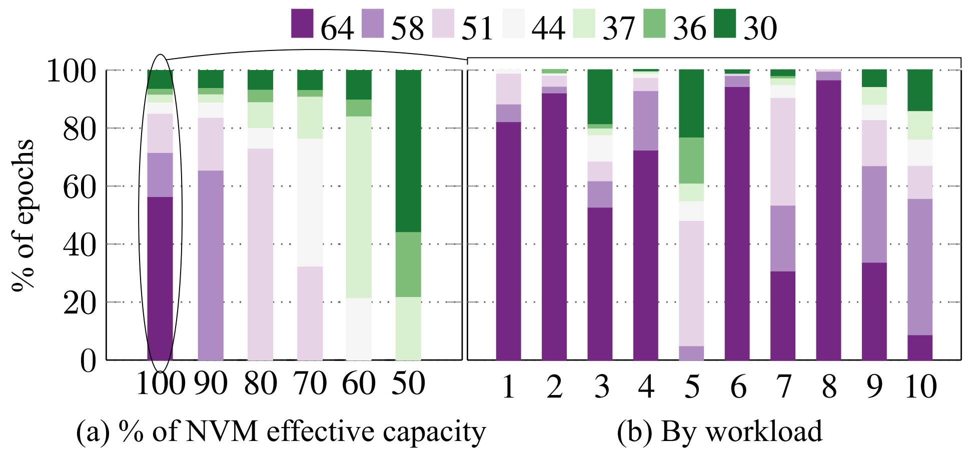


***Figure** 3.6: CA_RWR insertion policy: distribution of $CP_{th}$ achieving the best hit rates across execution epochs, vs. NVM part capacity (a). Uniquely for 100% NVM capacity, the same distribution, but for each of the 10 workloads (b).*

Let us consider the bar that represents the NVM cache with 100% capacity in Figure 3.6a. This is the same cache that was used in Figure 3.4, where we observed that the maximum hit rate is achieved with $CP_{th}$ values 58 or 64. However, Figure 3.6 reveals that these $CP_{th}$ values are not optimal throughout the entire workload execution. In 30% of the epochs, the optimal hit rate is attained with $CP_{th}$ values less than 58. Furthermore, this percentage varies greatly depending on the workload, reaching 96% in mix 5, see Figure 3.6b. In Figure 3.5, we demonstrated that these smaller $CP_{th}$ values are beneficial as they reduce BW to the NVM. This implies that a fixed $CP_{th}$ value may easily lead to both sub-optimal overall system performance and sub-optimal NVM lifetime. The impact of varying optimal $CP_{th}$ values becomes even more prominent as the cache loses effective capacity, see Figure 3.6a, because frames with higher capacities become more scarce.

For an adaptive $CP_{th}$ value selection mechanism, we propose CP_SD, a new insertion policy using Set Dueling [102] that reacts to both the changing workload behavior and the decreasing capacity of the NVM part. We propose to specialize some sets to use a fixed value of $CP_{th}$, from 30 to 64. Every value is tested on a group of N/32 sets, where N is the number of sets in the LLC. The rest of the sets follow the group of sets whose $CP_{th}$ brings optimal performance, the group of sets with the maximum number of hits in the previous epoch.

**CP_SD evaluation.** The red horizontal lines in Figures 3.4 and 3.5 indicate hit rate and BW of CP_SD, respectively. CP_SD achieves a hit rate equivalent to the best-case CA_RWR (with values of $CP_{th}$ 58 and 64), which is also comparable to the hit rate of the reference system BH. However, in terms of BW, CP_SD reduces the number of writes by a significant 83.4% compared to the BH cache and by 22.9% and 42% compared to CA_RWR for $CP_{th}$ 58 and 64, respectively. Besides, we perform these experiments varying the epoch size and our evaluation shows that 2M cycles achieves the best Set Dueling performance. This value is used for all evaluations in the following sections.

### 3.4.4 CP_SD_Th: CP_SD for both performance and lifetime

By using the CP_SD insertion policy it may happen that a very small difference in performance in an epoch determines the selection of a $CP_{th}$ value that produces a much larger number of BW to the NVM part. We thereby introduce CP_SD_Th: a variation that seeks a better tradeoff between performance and lifetime. It is based on selecting $CP_{th}$ considering not only the number of hits in LLC but also *the number of BW to the NVM part.*

We have not found a simple arithmetic function that combines both metrics to compute the Set Dueling winner, largely because their ranges of variability are very different and highly dependent on workload and NVM cache capacity. Alternatively, we will make a rule-based decision with two thresholds: i) *Th*, the maximum percentage of cache hits we are willing to sacrifice, and ii) *Tw*, the minimum percentage of NVM BW decrement we require to admit a performance loss. As usual, the rule for choosing $CP_{th}$ is applied at the beginning of each epoch by first looking for the value $i$ of $CP_{th}$ that achieved the maximum number of hits in the previous epoch. Then, the smallest value $j$ of $CP_{th}$ that satisfies the following inequalities is selected:

$$H(j) > H(i) * (1 - \frac{Th}{100}) \ \& \ W(j) < W(i) * (1 - \frac{Tw}{100}) \tag{3.1}$$

Where $H(x)$ and $W(x)$ are the number of hits and bytes written to NVM, respectively, in the sampler sets whose $CP_{th}$ was $x$ in the previous epoch.

**CP_SD evaluation by varying *Th* and *Tw*.** Our evaluation of Set Dueling with different values for the parameters *Th* and *Tw* shows that the sensitivity of the number of hits and BW to NVM to the parameter *Tw* is very low; therefore, Figure 3.7 only shows data for values of *Th* 0, 2, 4, 6, and 8% (different colors) keeping *Tw* = 5%. The different shapes represent different NVM capacities: the circles, triangles and squares correspond to NVM part capacities of 100, 90 and 80%, respectively. Both the number of hits and BW to NVM are normalized to BH with 100% NVM capacity.

The observed trend is similar for the three NVM capacities analyzed. Increasing the value of *Th* always produces a decrease in both the number of hits and BW to NVM. However, the relative decrease is much larger in BW, especially when the cache capacity decreases. For example, going from *Th* = 0 to *Th* = 8% for 80% NVM capacity, the number of cache hits decreases from 0.925 to 0.916 (1.0% reduction) while the number of writes decreases from 0.059 to 0.035 (40.7% reduction). However, the same *Th* variation for a 100% NVM capacity results in a 18.7% relative decrease in the bytes written to NVM.

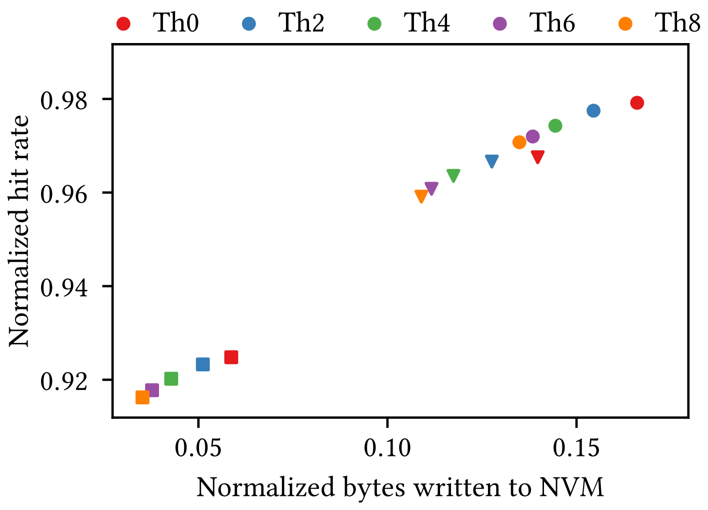


***Figure** 3.7: Hit rate and BW to NVM normalized to BH, for different* Th *[0-8%] and different NVM capacities: 100-90-80% (circles, triangles, and squares).* Tw *set at 5%.*

Note that the 8% limit on *Th* only produces a real loss of 1% in hits, since a) in many epochs no change in $CP_{th}$ is applied to decrease hits because in return there is not enough reduction in bytes written, and b) when the change is applied the decrease in hits will be between 0 and 8%. In addition, by construction, the decrease in bytes written can be much greater than the decrease in hits because the $CP_{th}$ is only changed in those epochs in which the decrease in bytes written is large and the decrease in hits is small.

## 3.5 Evaluation

This section evaluates the impact of compression-aware insertion policies on the lifetime and performance of a hybrid LLC, comparing them to the state-of-the-art proposals described in Section 3.2, see Table 3.2: CP_SD and CP_SD_Th refer to insertion policies introduced in Sections 3.4.3 and 3.4.4, respectively.

***Table** 3.2: Summary of tested insertion policies.*

| Name | Disabling granularity | Data Compression | NVM aware |
|---|---|---|---|
| BH | Frame | No | No |
| BH_CP | Byte | Yes | No |
| LHybrid | Frame | No | Yes |
| CP_SD | Byte | Yes | Yes |
| CP_SD_Th | Byte | Yes | Yes |

Section 3.5.1 introduces the experimental setup, including system specification, simulation infrastructure, and benchmark suites. Section 3.5.2 compares CP_SD insertion policies with state-of-the-art in terms of performance and NVM lifetime, showing how the

rule-based mechanism effectively trades performance in exchange for lifetime by tuning CP_SD *Th*. Sections 3.5.3, 3.5.4, 3.5.5, and 3.5.6 present sensitivity studies concerning to the ratio of NVM size vs. SRAM size, the impact of the coefficient of variation on system performance and NVM lifetime, the size of L2 (x2), and the increase in NVM access latency (x1.5); respectively. Finally, in Section 3.5.7 the cost of fine-grain disabling, i.e. the overhead of the byte fault map, is discussed, evaluating its impact by reducing the number of ways in the NVM part by an amount equivalent to such overhead.

### 3.5.1 Experimental setup

We use a 4-core system with private L1 (instructions and data) and L2 caches and a shared hybrid LLC, as detailed in Table 3.3. The hybrid LLC is non-inclusive and partitioned into four banks. The system uses a directory-based MOESI coherence protocol, and a crossbar that connects the private levels (L2) with the LLC banks and the directory.

***Table** 3.3: System specification.*

| | |
|---|---|
| Cores | 4, ARMv8, out-of-order (up to 8 inst/cycle), 3.5 GHz |
| Coherence Protocol | MOESI, directory distributed among LLC banks<br>64 B data block in all levels |
| L1 | Private, 32 KB D, 32 KB I, 4 ways, LRU<br>3-cycles load-use delay. Fetch on write miss |
| L2 | Private, 128 KB D, 128 KB I, 16 ways, LRU<br>11-cycle load-use delay. Fetch on write miss |
| Hybrid LLC | Shared, non-inclusive, 4 banks, 4MB/bank |
| | 4 SRAM ways, 28-cycle load-use delay (4-cycle D-array) |
| | 12 NVM ways, 32-cycle load-use delay (8-cycle D-array)<br>+2 cycles for decompression and block rearrangement<br>20-cycle data array write latency<br>Endurance: mean = $10^{10}$ writes, cv = 0.2 |
| Main Memory | 1 memory controller, DDR4<br>1 channel, 8GB/channel (1200 MHz) |
| NoC | Crossbar between the hybrid LLC banks and L2s. 32 B flits |

As for the simulation, we use two different infrastructures: a trace-driven simulator called HyCSim, see Chapter 5 for design space exploration and for figures in Section 3.4, and gem5 [83] for the detailed cycle-accurate full-system simulation presented in this Section. To estimate hybrid LLC latencies we use NVSim [33].

We adapted the forecasting procedure, which is introduced in Chapter 4, to the hybrid LLC scenario. It allows to accurately measure the impact of different insertion policies on the evolution over time of performance and capacity of the NVM part, taking into account the disabling of frames or bytes and the use of compression. BH and LHybrid are provided with frame-disabling to tolerate hard faults while BH_CP and CP_SD employ byte-disabling together with data compression. The forecasting procedure alternates between simulation and prediction phases. The simulation phase starts reading the NVM LLC state; for instance, in BH_CP and CP_SD such state is the fault map of every NVM frame, then it performs a full system simulation reporting several indexes of interest, e.g., the write rate on NVM frames, system IPC, and LLC hit rate. The prediction phase receives such write rates, computes the next $k$ NVM bitcells to become faulty, and update the fault map for the next simulation. In this work, the forecasting procedure advances in time until the NVM part loses 50% of its capacity, but there is no problem in reaching full depletion. The IPC evolution depicted in the figures of following sections is obtained at each simulation phase, computing the arithmetic mean of the IPCss of the mixes conforming the workload.

The experimental evaluation is made on the ten multi-programmed workloads shown in Table 3.4. Mixes are formed by randomly selecting applications from the SPEC CPU 2006 and 2017 suites, leaving out applications that do not show substantial memory activity [12, 56]. Fast forward is performed for 2 billion instructions, warm-up for 60 million cycles, and then 200 million cycles are simulated to collect statistics.

***Table** 3.4: SPEC CPU 2006 and 2017 mixes.*

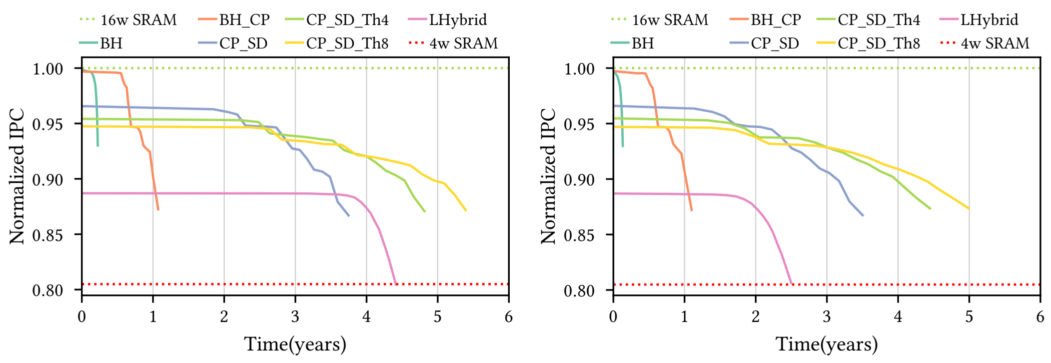
| | |
|---|---|
| mix 1 | zeusmp06 gobmk06 dealII06 bzip206 |
| mix 2 | hmmer06 bzip206 wrf06 roms17 |
| mix 3 | zeusmp06 cactuBSSN17 hmmer06 soplex06 |
| mix 4 | omnetpp06 astar06 milc06 libquantum06 |
| mix 5 | xalancbmk06 leslie3d06 bwaves17 mcf17 |
| mix 6 | lbm17 xz17 GemsFDTD06 wrf06 |
| mix 7 | cactuBSSN17 dealII06 libquantum06 xalancbmk06 |
| mix 8 | gobmk06 milc06 mcf17 lbm17 |
| mix 9 | xz17 astar06 bwaves17 soplex06 |
| mix 10 | GemsFDTD06 omnetpp06 roms17 leslie3d06 |

### 3.5.2 Performance vs. Lifetime

The solid lines in Figure 3.8a show the performance evolution over time of the proposed insertion policies for hybrid LLCs (CP_SD, CP_SD_Th), comparing them with BH, BH_CP and LHybrid. Dashed lines mark the upper and lower performance bounds. The upper bound corresponds to a a 16-way SRAM LLC, while the lower bound corresponds to a hybrid LLC whose NVM part is fully impaired (4w SRAM). Finally, we also show performance of a BH with compression (BH_CP) LLC. BH_CP uses compression and byte-disabling, but it is oblivious to the NVM wear due to writing and uses a global LRU-Fit replacement policy. In BH_CP, the victim frame is the one containing the LRU block from among those occupying frames with a size greater than or equal to the size of the block to be inserted, either in the SRAM or NVM.

Initially, in the first few months, the performance of BH_CP is similar to that of BH because both use a global, unconstrained LRU replacement algorithm. However, compression and byte disabling, even without compression-aware insertion and replacement policies, reduce the number of writes in the NVM part and manage to extend the lifetime of BH_CP by 4.8× with respect to BH. Still, the effective capacity of 50% is reached in 13 months, far short of the LHybrid 53 months.

CP_SD, the performance-optimized configuration, manages to delay the 50% capacity loss to 45 months, multiplying BH lifetime by 16.8×. This increase in lifetime is achieved



(a) cv = 0.2

(b) cv = 0.25

***Figure** 3.8: Performance evolution until the NVM part reaches 50% effective capacity for different CP_SD Th, for default parameters (cv* = 0.2*) (a), and for cv* = 0.25 *(b).*

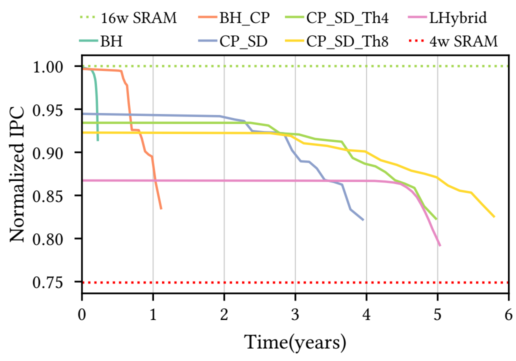
at the cost of a performance loss of only 3.3% at the beginning of the cache lifetime. This performance level remains almost unchanged beyond two years. From this point on, the NVM part starts to gradually lose capacity, which translates into a gradual drop in performance. Compared to LHybrid, CP_SD reaches the 50% capacity 8 months earlier but always maintains a significant difference in performance, especially in the long initial stage where it reduces the performance loss compared to the upper limit (dotted green line) from 11.2% of LHybrid to only 3.3%.

As introduced in Section 3.4.4, CP_SD can be further tuned to trade performance in exchange for lifetime and vice versa. Figure 3.8a shows the results for $Th$ 4 and 8%, keeping $Tw = 5$. CP_SD_Th4 and CP_SD_Th8 achieve 28% and 44% increase in lifetime compared to CP_SD, in exchange for 1.1% and 1.9% performance degradation, respectively. Compared to LHybrid, CP_SD_Th4 and CP_SD_Th8 achieve 9% and 22% more lifetime while keeping 7.6% and 6.8% higher performance, respectively.

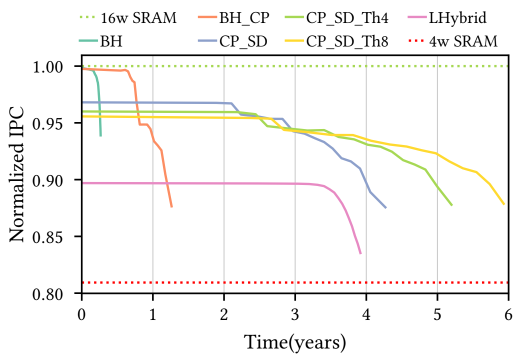
### 3.5.3 SRAM-NVM proportion variation

We analyze the behavior of the hybrid LLC by increasing asymmetry between the sizes of the SRAM and NVM. Specifically, Figure 3.9a shows hybrid LLCs with a 3-way SRAM and a 13-way NVM part. The decrease in the number of SRAM ways has little impact on BH and BH_CP because block insertion and replacement do not depend on this parameter. The original 12-way NVM wears similarly in this new 3/13-way configuration. The only additional performance degradation occurs due to the loss of capacity of the new NVM way.

In LHybrid, the SRAM part acts as a read-reuse detector for the NVM part. By decreasing the number of SRAM ways, less read reuse is detected, thereby resulting in fewer block insertions to the NVM part. This translates into a 14% longer lifetime and a 2.2% lower performance compared to the 4/12 configuration. For CP_SD-based policies, increasing the SRAM/NVM asymmetry also means a slight increase in lifetime (5.5%, 3.4%, and 7.4%) and a slight drop in performance (2.2%, 2.1%, and 2.6%), for CP_SD, CP_SD_Th4, and CP_SD_Th8, respectively.



(a) NVM-SRAM proportion variation



(b) L2 size = 256KB

***Figure** 3.9: Performance evolution until the NVM part reaches 50% effective capacity for default parameters and different CP_SD Th, varying the NVM-SRAM proportion (a), and increasing L2 size to 256 KB (b).*

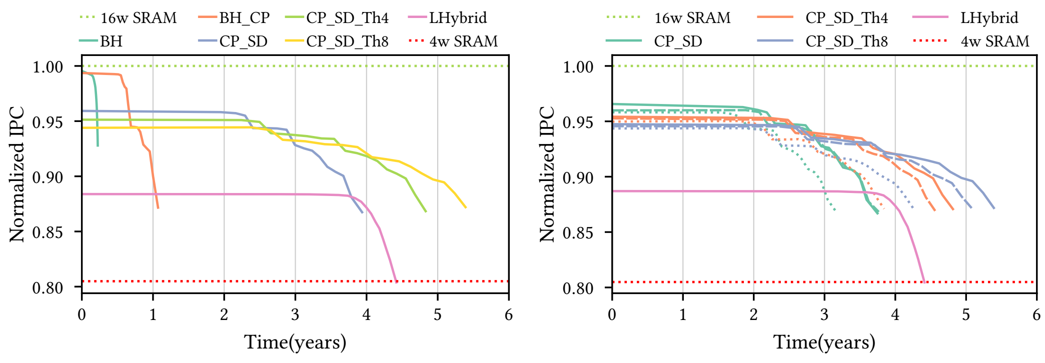
### 3.5.4 Impact of cv on performance and lifetime

We model the endurance of NVM memory bitcells using a normal distribution, see Section 1.1.4. The coefficient of variation *cv* of this distribution reflects the memory manufacturing variability. This parameter has a significant impact on the evolution of the LLC capacity. A higher coefficient of variation implies a larger dispersion in the number of writes supported by each cell. Consequently, the first faults occur earlier, thereby impacting frame- and byte-disabling techniques.

In Figure 3.8b we have repeated the experimentation assuming a higher manufacturing variability, changing the coefficient of variation *cv* from 0.20 to 0.25 and keeping the mean $\mu = 10^{10}$ constant. The lifetime of frame-disabling caches is drastically reduced as the coefficient of variation increases. The time to reach 50% capacity goes from 2.7 months to 1.6 months for BH and from 53 to 30 months for LHybrid. However, the impact on the lifetime of the models with byte-disabling is much smaller: for BH_CP, it remains the same, for CP_SD it drops from 45 to 42 months, for CP_SD_Th4, it drops from 58 to 53 months, and for CP_SD_Th8 it drops from 65 to 60 months. Consequently, CP_SD-based policies manage to significantly improve both performance and lifetime over LHybrid as *cv* grows: CP_SD, CP_SD_Th4, and CP_SD_Th8 achieve 1.4×, 1.8×, and 2× greater lifetime while maintaining 8.9%, 7.6%, and 6.8% greater performance, respectively.

### 3.5.5 L2 size sensitivity

Figure 3.9b shows performance evolution when L2 size is increased from 128 to 256 KB. Increasing the L2 size means increasing the overall system performance. Besides, it also means that the L2 can filter more write operations from the hybrid LLC, which translates into a slight increase in lifetime. Compared to the systems in Figure 3.8a, lifetime increases 19%, 18%, 14%, 8%, and 10% for BH, BH_CP, CP_SD, CP_SD_Th4, and CP_SD_Th8, respectively. On the contrary, the lifetime of LHybrid decreases by 11%. As already mentioned, in LHybrid, a block must experience a hit in the SRAM ways before being inserted in the NVM ones. When the L2 capacity is increased, the SRAM activity and the number of block-fills decrease, so the blocks spend more time in the SRAM ways. The more time a block is present in the LLC, the higher the probability of it being hit and detected as a loop-block. Detecting more loop-blocks results in an overall increase in the write rate to the NVM part and, thereby, a lifetime reduction.



(a) 1.5× NVM latency (b) Equalizing costs

*Figure 3.10: Performance evolution until the NVM part reaches 50% of effective capacity, increasing 50% the NVM data array latency (a), and equalizing costs of CP_SD systems with LHybrid (b).*

### 3.5.6 NVM latency sensitivity

NVM technology and system integration may have various optimization targets. As a result, the NVM latency might vary significantly. Figure 3.10a shows results for an NVM latency equal to 1.5× the original one, i.e. the NVM data array read latency is increased from 8 to 12 cycles. As expected, policies that insert more aggressively on the NVM part are more affected by increased latency than those that insert more conservatively. For instance, compared to Figure 3.8a, the performance at the beginning of CP_SD, CP_SD_Th4, CP_SD_Th8, and LHybrid decreases by 0.7, 0.3, 0.4, and 0.4%, respectively. This small drop in performance translates into a slight reduction in the NVM write rate, and these configurations experience a slight increase in lifetime. However, overall, there is no drastic change in the hybrid LLC performance and lifetime.

### 3.5.7 Overhead analysis & Equalizing costs

In the previous sections it has been shown that insertion policies tailored to compression and byte disabling improve the state of the art in both performance and lifetime, and achieve this even with a higher read latency due to rearrangement and decompression, see Section 3.3. But of course, this is at the cost of a non-negligible storage overhead. It is therefore necessary to re-evaluate the comparison, using the same total storage in the systems without and with compression.

All evaluated configurations employ SECDED protection, able to point out the faulty bitcell and disable the corresponding region, see Section 3.3. Regarding the metadata overhead, BH and the state-of-the-art (LHybrid, TAP) are provided with frame-disabling and hence require one bit per NVM frame. BH_CP and CP_SD, similar to L2C2, see

Chapter 2, need a fault map to disable at byte granularity: one bit per NVM byte. Compared to LHybrid, CP_SD incurs a storage overhead of 8.6%, i.e., 12.3% of the NVM data array.

Hence, we now analyze the performance and lifetime of CP_SD, CP_SD_Th4 and CP_SD_Th8 with a similar storage cost to LHybrid. We thereby reduce the number of NVM ways of these caches from 12 to 11 and 10, which results in 1.8% higher and 5.2% lower storage cost than LHybrid, respectively.

The solid, dashed, and dotted pattern lines in Figure 3.10b show the data from caches with 12, 11, and 10 NVM ways, respectively. All CP_SD configurations decrease their performance and lifetime when the number of ways is reduced. Nonetheless, the normalized IPC in the initial phase of the cache lifetime is in all configurations significantly higher than that of LHybrid. The CP_SD_Th8 cache with 10 NVM ways, with a 5.2% lower storage cost than LHybrid, manages to increase the normalized IPC of LHybrid by 6.4% during the first two years and maintains a higher IPC throughout the whole life of the cache.

## 3.6 Concluding remarks

Hybrid LLCs bridge the performance and capacity gap between the high-performance SRAM and high-capacity NVM LLC designs. Existing hybrid LLC proposals particularly optimize for LLC lifetime by only conservatively inserting cache blocks into the NVM ways. These lifetime-focused optimizations significantly reduce the LLC performance.

In this chapter, we leverage that 78% of the total LLC blocks are compressible to some extent and thus propose fault-aware policies to smartly steer cache blocks into the NVM or SRAM ways by analyzing both the cache block read-/write-reuse behavior and its compressed size. We use Set Dueling to identify the best-performing compression threshold, $CP_{th}$, depending on the workload behavior and the NVM capacity. Our proposed insertion policy can be further tuned to trade performance in exchange for NVM lifetime by adjusting the NVM write rate with a rule-based mechanism.

Our evaluations show that our insertion policies with Set Dueling nearly achieve the performance of a SRAM cache with the same associativity while improving lifetime by 17× compared to a hybrid NVM-unaware LLC. For a fair comparison, we adapt state-of-the-art hybrid LLC insertion policies to a fault-aware environment. Our design outperforms the state-of-the-art by 9% while attaining a comparable lifetime. Besides, the rule-based mechanism can achieve, for instance, 9% and 22% more lifetime than LHybrid while still outperforming it by 7.6% and 6.8%, respectively.

# Part II

# Methodological improvements for NVM-based LLCs

*A man must have a code.*

Detective Bunk Moreland, The Wire.

# 4

# Forecasting lifetime and performance of NVM-based LLCs

*Although different approaches are used in the literature to analyze the degradation of a NV-LLC, none of them allows to study in detail its temporal evolution. This chapter proposes a forecasting procedure that combines detailed simulation and prediction, allowing an accurate analysis of the impact of different cache control policies and mechanisms (replacement, wear-leveling, data compression, etc.) on the temporal evolution of the figures of merit, such as the effective capacity of the NV-LLC or the system IPC.*

*This forecasting procedure combines cycle-accurate simulations with predictions. The simulations receive the healthy state of the NVM-based LLCs and report several indexes of interest, e.g., the write rate on NVM frames, system IPC, and LLC hit rate. The prediction phases receive such write rates and computes the next NVM bitcells to become faulty. As a result, such a forecasting procedure enabled us to comprehensively analyse different microarchitectual optimizations and content management policies in Chapters 2 and 3.*

## 4.1 Introduction

NVMs can potentially replace large SRAM memories such as the LLC. Despite their many advantages, non-volatile memories face a critical challenge related to endurance, often referred to as the "wear-out" problem. This challenge stems from the limited number of write operations that NVMs, such as ReRAM, PCM, or STT-MRAM can withstand before their performance and reliability begin to degrade [7,14,65,74,77,101,106,107,130,137]. The repeated write operations that these memories endure during regular operation gradually lead to material fatigue, resulting in a deterioration of the memory cells and an eventual data corruption.

The essence of the problem is as follows. NVM bitcells age with writes. And, as memory degrades, performance and write rates change as well. However, its detailed simulation, cycle by cycle, requires a time that would far exceed the lifetime of the system under study.

Focusing on the LLC, its degradation leads to an increase in the miss rate, which results in a loss of performance which in turn results in a decrease of the NV-LLC write rate. Let's quantify how the write rate per frame may change as the NV-LLC degrades. Figure 4.1 shows the average write rate per frame in a 16MB, 16-way frame-disabling NV-LLC at various aging stages (see Section 2.5.1 for the simulated system details). Each bar depicts the average write rate of all frames belonging to a given group of sets, namely the sets with *A* alive frames in a degraded NV-LLC with 90%, 75% and 50% effective capacity, respectively.

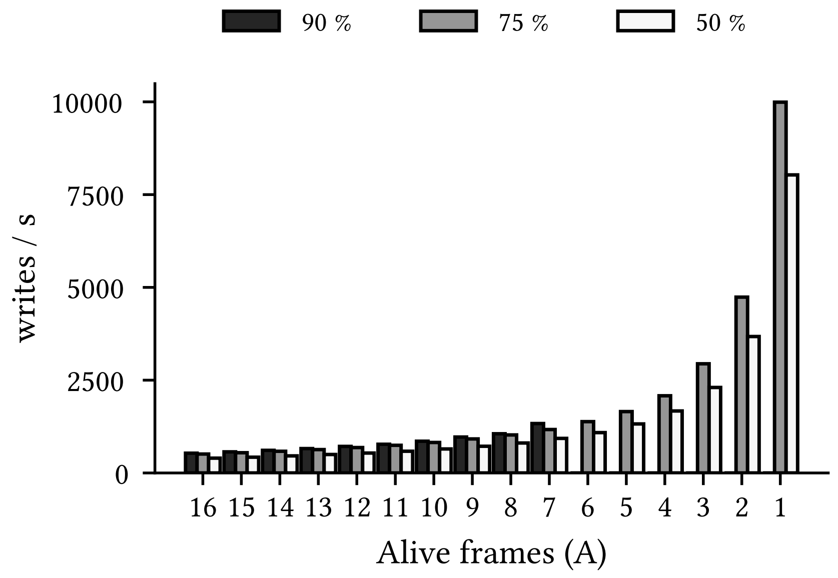


***Figure** 4.1: Average write rate per frame in sets with A alive frames as a function of capacity (90%, 75%, and 50%).*

At 90% capacity, all sets have between 16 and 7 alive frames. However, when the capacity is reduced to 75%, more degraded sets appear, which only have between 6 and 1 alive frames. Regardless of the capacity, as *A* decreases, the write rate per frame increases noticeably. This increase in write rate has two causes: i) the miss rate increases in the sets with fewer alive frames and therefore those sets experience a higher write rate, and ii) the write rate per set will be spread over fewer alive frames. By contrast, when considering the reduction in capacity from 75% to 50%, a decrease in the write rate per frame is observed for any value of *A*, which is due to a noticeable decrease in system performance.

Furthermore, this non-uniform degradation may affect differently the threads sharing the NV-LLC, selectively reducing the IPC of some of them and changing the pattern of writes in the entire NV-LLC. The existence of compression further complicates the modeling, as the data set referenced by each thread may have different compression capabilities that will wear cache bytes unevenly. In short, in order to capture the complexity of all these interactions a comprehensive assessment procedure is needed.

**Contributions.** Accurately addressing this feedback between degradation and performance loss over the lifetime of NV-LLCs is crucial to show and prove the microarchitectural enhancements introduced in Chapters 2 and 3. We introduce a forecasting procedure to estimate the evolution over time of any metric of interest linked to the NV-LLC (effective capacity, miss rate, IPC, energy, etc.), from the time it starts operating until its storage capacity is exhausted.

Forecasting relies on a sequence of epochs that sample the lifetime of the NV-LLC. Each epoch starts with a performance simulation and ends with an aging prediction. The *performance simulation* is carried out with cycle detail on a snapshot of the NV-LLC at a particular aging stage and obtains performance metrics (miss rate, IPC, etc.) and in particular, all the write rate statistics needed to feed the aging prediction. The *aging prediction* removes from operation the bytes or frames that die, according to the bitcell endurance model, the NV-LLC organization, and the write rate statistics received. At the end of each aging prediction phase, a new NV-LLC snapshot is generated, with lower capacity than the previous one.

Thus, performance and capacity forecasting considers the interaction between the workload and the non-uniform degradation of the NV-LLC in its multiple dimensions (bank, set, way, byte). It can be applied to a wide range of NVM-based main memory or NVM-based cache designs, although in this chapter we have focused on L2C2 and related alternatives, considering replacement and operation with compressed blocks and degraded frames under different redundancy schemes. Of course, performance or capacity forecasts are useful for research purposes, but also can be an industry tool to estimate the life cycle of an NV memory and provide customers with a clear commitment to lifespan and performance. The code is available so that anyone can use it for research purposes [36].

The rest of the chapter is organised as follows. Section 4.2 reviews literature in aging models and studies of NVMs. Section 4.3 introduces the forecasting procedure, which is conducted both on systems provided with frame disabling and on systems provided with byte disabling and data compression. Section 4.4 evaluates and validates the forecasting procedure. Section 4.5 further discusses the applicability of forecasting procedures to other computer architecture problems. Finally, Section 4.6 concludes this study.

## 4.2 Related work

Previous work on aging and degradation of NV-LLCs often highlights the difficulty of accurately modeling aging in NVMs and its effects on performance. In the absence of a standard procedure, practical solutions have been proposed, designed to assess specific aspects of one or another mechanism.

A first group of papers related to the evaluation of reliability improvement in NVM-based main memories or caches, focuses exclusively on measuring either the reduction in the number of writes or their variability [31, 96, 121]. For instance, Wang et al. compare wear-leveling mechanisms in NV-LLCs by calculating the elapsed time from startup to the first bitcell fault [121]. Such cache lifetime is computed by dividing the maximum number of writes supported by a bitcell by the number of writes per unit time (write rate) on the cache line that accumulates the most writes. The procedure consists of a single cycle-accurate simulation to record write variability, followed by an aging prediction that assumes such variability to be constant throughout the life of the cache. This procedure is simple, fast and allows the production of performance metrics such as the number of IPC, but does not consider the manufacturing variability in bitcell endurance. More importantly, it also does not allow to calculate the time evolution of the capacity or performance in degraded mode of operation, in which cache frames are progressively lost.

A second group of works, focused on extending the main memory lifetime, already incorporate process variability, modeling bitcell endurance by means of a normal probability distribution [60, 63, 110, 112, 133].

Ipek et al. [60] and Seong et al. [112] assume that writes are spread evenly across the main memory. Their quality metric is the number of writes the memory can receive until the first unrecoverable fault occurs on any of its pages [112] or until the memory loses all its capacity (each page is deactivated when it reaches its write limit) [60]. They do not relate the number of writes to the time elapsed, and therefore do not need to simulate any application. Yoon et al. propose the same quality metric [133], but assume that the page write rate is constant, thus expressing memory lifetime in elapsed time, rather than number of writes.

Schechter et al. [110] and Jadidi et al. [63] simulate a workload on a system whose main memory has no faulty cells. The former to obtain frequency of writes and the latter to obtain traces of memory accesses. Schechter et al. evenly distribute the number of writes among all alive pages and calculate the bitcell that will fail first [110]. When the number of faulty cells in a page reaches a threshold, the page is deactivated and its writes are distributed evenly among the remaining alive pages. Jadidi et al. use the trace of writes to main memory to accumulate the number of writes to each bitcell, deactivating them when they reach their maximum number [63]. The simulation is repeated several times, until the memory loses half of its capacity. They measure the lifetime by counting the number of times the trace is reinjected. Now, from the execution time of the detailed simulation that produced the trace they can calculate a lifespan in terms of elapsed time, but always assuming that performance does not vary with memory degradation.

In summary, no procedure capable of accurately estimating the simultaneous degradation of capacity and performance over time has been proposed to date.

## 4.3 Forecasting procedure

This section describes a procedure to forecast the capacity and performance evolution of an NV-LLC through time, from its initial, fully operational condition, until its complete exhaustion. The forecast is driven by a detailed, cycle-by-cycle simulation of a workload, and the maximum number of writes supported by bitcells is modelled by a normal distribution, as stated in Section 1.1.4.

The forecasting procedure determines the alive byte configuration in discrete steps of capacity loss, which we call epochs. An epoch starts with a detailed Simulation phase, where performance and write rate measurements are extracted, and continues with a Prediction phase, where the corresponding memory region that fails is disabled depending on the disabling granularity, and the remaining number of writes of those bytes that are still alive is updated.

To the best of our knowledge, this is the first NV-LLC capacity and performance forecasting procedure proposed so far.

### 4.3.1 Data structures supporting the forecasting procedure

For each byte of the data array it is necessary to keep track of two key attributes, namely the number of remaining writes (RW) per-byte and the write rate (WR) per-byte. These attributes are represented in two data structures, called maps and abbreviated as *RW map* and *WR map*, respectively.

**RW map**. Each entry of *RW map* holds the number of remaining writes $rw_{ijk}$ of byte $B_{ijk}$ (set $i$, way $j$, byte $k$), see Figure 4.2a. *RW map* is initialised according to the statistical endurance model of the memory technology used, as in [28, 44, 60, 110, 112, 133], see Section 1.1.4.

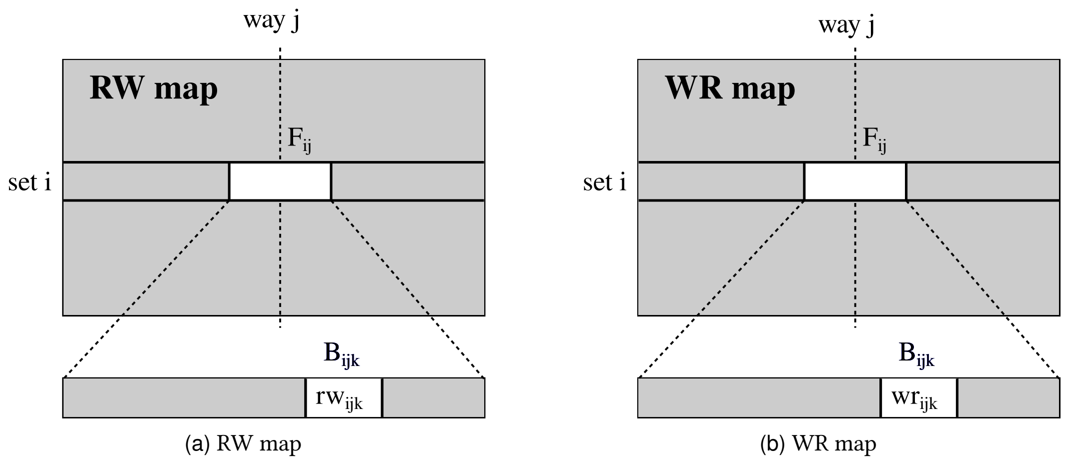


***Figure** 4.2: Per-byte remaining writes (RW) (a) and write rate (WR) (b) maps.*

Once the *RW map* is initialized, it would be sufficient to simulate the NV-LLC with the desired workload and update the map on each write, subtracting in all the alive bytes of the frame being written. When any byte of the cache reaches its maximum number of writes ($rw_{ijk} = 0$), the corresponding cache region is disabled, the whole frame with frame disabling or the single byte with byte disabling. Then, the simulation would continue with the degraded system.

The simulation should be detailed, cycle-accurate, so that the progressive degradation is reflected in the miss rate and write rate of the remaining healthy regions. However, this naive approach is not feasible, since at detailed simulation speed only a few milliseconds of forecast could be attained.

**WR map**. An alternative approach, which is nearly as accurate, but with lower simulation cost is the following. After a suitable simulation time we write down in a *WR map*, the write rate per byte $wr_{ijk}$, see Figure 4.2b. On the assumption that these per-byte write rates remain constant as long as no further byte is disabled, we can compute the *predicted lifetime (PLT)* of each byte $B_{ijk}$ as:

$$PLT(B_{ijk}) = \frac{rw_{ijk}}{wr_{ijk}}$$

We can use $PLT$ to predict the next byte that becomes faulty.

## 4.3.2 Basis of the forecasting procedure

The lifetime of an NV-LLC can be forecast using the procedure outlined with black lines in Figure 4.3. The *RW map* is first initialized taking samples from a normal statistical distribution of the maximum number of writes a bitcell can endure. Forecast then proceeds through successive *epochs*, which consist of a Simulation phase followed by a Prediction phase.

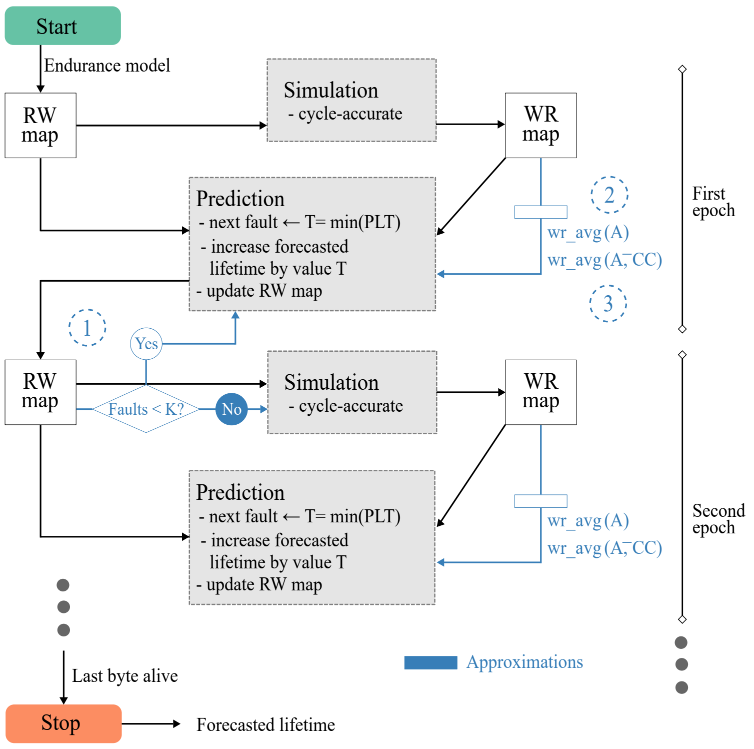


***Figure** 4.3: Forecasting procedure diagram. Basic procedure in black, approximations in blue.*

The Simulation phase requires the development of a microarchitectural LLC model that allows to dynamically configure a different associativity in each set and, if applicable, a different number of bytes per frame. The Simulation will thereby consider the cache regions that are still alive according to the *RW map*, run the workload for a suitable number of cycles, compute the write rates in each alive byte, and finally update the *WR map*.

The Prediction phase combines the values of both maps to calculate $PLT(B_{ijk})$, selecting the byte with the lowest remaining lifetime, $t = min(PLT(B_{ijk}))$. The prediction consists of advancing the forecasted lifetime by exactly that value $t$. To do so, it is sufficient to subtract from the number of remaining writes in each byte of the cache, the number of writes that would have occurred in that byte in a time $t(\forall\ ijk : rw_{ijk} = rw_{ijk} - t * wr_{ijk})$. In this way the next simulation will be performed with the corresponding region disabled, cache frame or byte, so that the behavior of the LLC will take into account the degradation experienced in the data array.

Forecast advances through single-prediction epochs until all bytes in the cache are disabled. Each epoch adds the variable time $t$ to the NV-LLC lifetime, which depends on the initial *RW map* and the write rate variation. Although the Prediction phase is computationally very light, this alternative approach still requires as many simulations as there are bytes in the cache, and it is also not affordable considering the runtime required for a detailed simulation.

To decrease the number of simulations we propose an approximate procedure that extends the forecast duration within each epoch. This approximate forecasting procedure acts as follows. In each epoch, the simulation phase does not change: it receives an *RW map* and obtains the corresponding *WR map*. However, the prediction phase has an *extension* of $K$ consecutive predictions, corresponding to the failure of $K$ bytes. After every prediction step the *RW map* is updated; see ① in Figure 4.3.

The challenge now is that, as bytes die during the multiple-prediction epoch, the values of the *WR map* may not reflect the effect of the progressive degradation of the NV-LLC during the epoch.

When a cache byte dies there may be a tiny decrease in hit rate and system performance, which may result in tiny changes of the byte write rate across all cache frames. Our model does not take this reduction into account during the Prediction phase within each epoch. However, if we focus on a shrinking cache set, i.e. one in which a byte has just been disabled, the new write rate in the frames of that set can increase significantly. This effect is evident with frame disabling, see Figure 4.1, but occurs equally with byte disabling. Consequently, to increase the epoch extension without introducing significant error, a model is needed to approximate new write rates as bytes fail during prediction.

Without loss of generality, a uniform distribution of writes among cache sets is assumed in this paper; see Section 4.4.3 for a more general discussion. Accordingly, the write rate on the bytes of a cache set whose *health state* has just degraded after a prediction step can be computed by the average write rate of all the bytes belonging to the sets that *were already* in that degraded health state during the simulation. As will be seen below, the health state of a cache set is defined differently for frame- and byte-disabling caches.

### 4.3.3 Approximate forecasting procedure for frame disabling

In frame disabling, all bytes in a frame receive the same write rate, and it matches the write rate in the frame. Therefore, the *WR map* stores information at frame granularity.

Under the assumption of a uniform distribution of references across sets, for NV-LLCs with frame disabling, the health state of a set can be defined simply as its $A$ number of alive frames, with $A$ between one and the initial associativity.

At the end of a Simulation phase, $wr\_avg(A)$, the average write rate per byte in sets with $A$ alive frames, is computed from the *WR map*; see ② in Figure 4.3. Thus, during the

Prediction phase the write rate applied to the bytes of a frame changes as the health state of its set changes. That is, while a set has $A$ alive frames, the prediction calculations age its bytes with $wr\_avg(A)$, but when one of them dies, the aging will be performed with $wr\_avg(A-1)$.

Note that in the Prediction phase, after disabling a certain number of frames, sets with a value of $A$ not yet simulated may appear. For instance, let us focus on the black distribution of write rates per frame we showed in Figure 4.1. It corresponds to the Simulation phase of an epoch that starts with 90% effective capacity. In that epoch, the Prediction phase handles sets with 7 or more alive frames. But before reaching $K$ predictions a byte belonging to a set with $A = 7$ may die, appearing a new health state, that of the sets with $A = 6$ for which there are no available write rate data yet. To cope with these cases, we can stop the prediction, thus ending the epoch prematurely and starting a new simulation. Alternatively, to keep low the number of simulations, we can continue the prediction, also allowing some more error and apply the previous value $wr\_avg(7)$. In this work, we will adopt this second approach.

### 4.3.4 Approximate forecasting procedure for byte disabling and data compression

Unlike frame disabling, in a NV-LLC with byte disabling and data compression, such as L2C2, a write to a frame does not always imply a write to all the bytes of the frame and therefore the write rate to the bytes of a frame is lower than the write rate to the frame. The wear-leveling mechanism ensures an even distribution of writes among the alive bytes of a frame. Consequently, during prediction, we can assume that the write rate on all alive bytes of a frame is equal, and is calculated as the average of the write rates on all of them.

Moreover, a fault in one or more bytes of a L2C2 frame does not preclude storing blocks, as long as their compressed size is appropriate. Now the health state of its sets is more diverse than in frame disabling: at any given time there are not only alive and dead frames, but frames with a very diverse range of effective capacities.

The number of faulty bytes in a frame limits the compression encodings (CEs) it can accommodate. A frame with a certain effective capacity is associated with a compression class (CC) if it can accommodate compressed blocks of size CC or smaller. For example, a frame with 3 defective bytes has an effective capacity of 61 bytes, which accommodates blocks of any compression encoding but those of size 64 bytes (see Table 2.1); it is thereby associated with CC = 58.

In this context, the prediction of write rate per byte is more complex. For example, let's consider a set that has only one frame of CC = 64. All non-compressible blocks will end up in that single frame, which can become a hot spot for writes within the set. But, in another cache set with a majority of frames with CC = 64, the write rate of the set will be distributed in a substantially equal way among frames.

With this, our Prediction phase will assume that the write rate a byte receives depends on the CC of its frame as well as on the CCs of the rest of the frames in the same cache set. Therefore, now the health state of a set is abstracted as a 12-tuple $\bar{A}$. It aggregates the compression classes to which each frame belongs to. For instance, a set with tuple $\bar{A}$ = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 15) has one frame with CC = 58 and 15 frames with CC = 64.

Thus, during prediction, the aging write rate to consider for the bytes of a given frame $F_{ij}$ will depend on its compression class CC and the health state (tuple $\bar{A}$) of the set that contains $F_{ij}$: $wr\_avg(\bar{A}, \text{CC})$.

More specifically,

$$\begin{aligned} wr\_avg(\bar{A}, \text{CC}) &= average(wr_{ijk}) \\ &\forall\, ij \mid \text{i) } F_{ij} \in \text{ set with tuple } \bar{A} \\ &\qquad \text{ii) } F_{ij} \in \text{ compression class CC} \end{aligned}$$

Each time a byte is disabled in a frame $F_{ij}$, CC of the frame and $\bar{A}$ of the set are recomputed. Thereafter, $wr_{ij}$, the aging write rate of $F_{ij}$ with compression class CC, is approximated by $wr\_avg(\bar{A}, \text{CC})$; see 3 in Figure 4.3.

As in frame disabling, as faults are predicted in succession, sets with a tuple value $\bar{A}$ not yet simulated may appear. For instance, suppose a set with the same 12-tuple as before: one frame associated with CC = 58, and 15 frames associated with CC = 64:

$$\bar{A}_1 = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 15).$$

Suppose a byte of one of the fifteen frames with CC = 64 fails during the Prediction phase. Now the tuple modeling the set becomes:

$$\bar{A}_2 = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 14).$$

But if in the epoch Simulation no $\bar{A}_2$ tuple was tracked, the values of $wr\_avg(\bar{A}_2, -)$ are unknown. As in frame disabling, in this work we chose to continue the prediction, tolerating some more error and using for that set the previous values of $wr\_avg(\bar{A}_1, -)$ as an approximation of $wr\_avg(\bar{A}_2, -)$.

## 4.4 Evaluation

In order to validate the forecasting procedure, it would be necessary to contrast its projections with data from the operation of real NV-LLCs as they age with a known workload. But unfortunately, there is no such information in the public literature. Therefore, in this section we provide tests of the correctness of the assumed hypotheses as a function of the number of epochs employed, evaluating the tradeoff between accuracy and time spent in the forecasting procedure. Finally, we outline alternatives for situations in which some underlying assumptions are not met.

The followed methodology, system configuration, and compared systems in this section are the same as in Chapter 2. Please refer to Section 2.5 should you need further details.

### 4.4.1 Validation

As discussed in Sections 4.3.3 and 4.3.4, the main source of forecast inaccuracy lies in the Prediction phase, where it is necessary to approximate the write rate of health states that have not yet appeared in the Simulation phase. Of course, using epochs of small extension implies low approximation and can improve the quality of the forecast, but at the same time it increases computational cost.

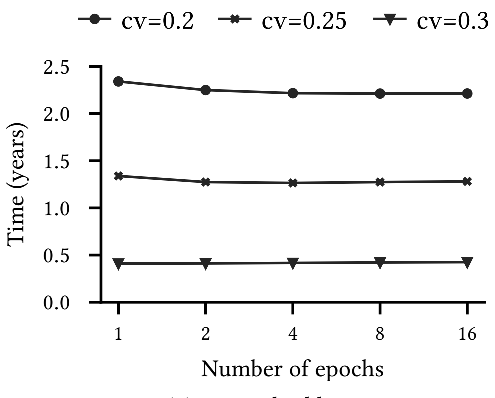
To explore this tradeoff between quality and cost, several experiments have been performed, using epochs of different extension in each experiment, which predict a certain NV-LLC degradation. Specifically, we predict how much time elapses until 50% of the NV-LLC, $T_{50C}$, degrades. A 50% capacity degradation is a common case study [63, 110, 133], and in our experiments we will also focus on it, but any other percentage, including 100%, corresponding to total degradation, could also be used.

A different number of epochs of constant extension is used in each experiment. The epoch extension is the number $k$ of consecutive predictions disabling frames or bytes, depending on the NV-LLC model, and is calculated by simply dividing 50% of the cache size, measured in frames or bytes, by the number of epochs.

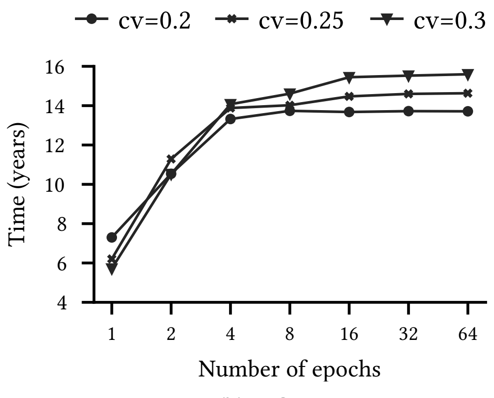
The Y-axes in Figure 4.4a and Figure 4.4b shows $T_{50C}$ as a function of the number of epochs for frame disabling and L2C2, respectively; built with bitcells of different manufacturing variabilities.

As can be seen, the forecast of $T_{50C}$ converges as the number of epochs increases for all coefficients of variation. Using a number of epochs greater than or equal to 8 and 16, $T_{50C}$ varies less than 0.8 and 1.1% for frame disabling ($k$=16384 frames) and L2C2 ($k$=524288 bytes), respectively.

Previous works performs a single simulation from a fully operational NV memory to obtain the write rate data [60, 63, 110, 112]. From this data, they compute the time at which a bitcell dies, and then recalculate the write rate analytically. In this sense, this methodology is similar to ours when a single epoch is used. But, as Figure 4.4 shows, in both cases but specially for compression, using a single epoch incurs in a non-negligible error.



(a) Frame disabling



(b) L2C2

***Figure** 4.4: Forecasted $T_{50C}$ (in years) as a function of the number of epochs for frame disabling and L2C2 caches. Three coefficients of variation are employed: cv = 0.2, 0.25, and 0.3.*

Finally, in order to prove that different *RW maps* do not lead to inconsistent results, five different random seeds have been used. The seeds are used to initialize different *RW maps* for the three values of *cv* and forecast is performed for all of them. Again, the convergence metric is $T_{50C}$ in a 16-epoch forecast of an L2C2. The standard deviation of the different forecasted times is below 2% of the arithmetic mean.

### 4.4.2 Computational cost

In the following we provide the computational cost of the most expensive procedure, the one related to the forecasting of a byte-disabling LLC, together with actual time measurements.

The computational cost of the Simulation phase comes from the execution of the gem5 simulator. It depends on the number of mixes, $M$, used as workload and the number of cycles, $Cy$, simulated for each mix. Thus, using the same input parameters the simulation cost does not depend on the number of epochs.

The computational cost of the Prediction phase is proportional to the epoch extension, $K$, and to the size of the cache in bytes, $C$. Indeed, on the one hand, the cost is proportional to the number $K$ of bytes to be disabled, i.e., the epoch extension. On the other hand, it is also proportional to the size of the cache in bytes, $C$, since to predict the death of a byte it is necessary to scan the entire cache to find the byte $B_{ijk}$ whose $PLT(B_{ijk})$ is the minimum.

Thus, to forecast the evolution of an L2C2 until it loses a given fraction $f$ of its number of bytes $C$, $E = f\frac{C}{K}$ epochs are needed. Putting all together, the total forecasting cost of a fraction $f$ of capacity $C$, with $E$ epochs, $M$ mixes and $Cy$ cycles is:

$$\begin{gathered} Forecasting\,cost(f, C, E, M, Cy) = E \cdot cost(Sim + Pred) = \\ f_1(f \cdot \frac{C}{K} \cdot M \cdot Cy) + f_2(f \cdot \frac{C}{K} \cdot K \cdot C) = f_1(E \cdot M \cdot Cy) + f_2(f \cdot C^2) \end{gathered} \tag{4.1}$$

where functions $f_1()$ and $f_2()$ depend on the details of the server hardware carrying out the forecasting. In summary, the computational cost of the forecasting procedure is linear with the number of epochs and quadratic with the size of the cache. However, as we will see from the experimental data, the value of $f_2()$ is much smaller than that of $f_1()$.

Table 4.1 shows the maximum elapsed times, broken down into Simulation and Prediction phases, for a prediction reaching up to 50% degradation of L2C2 capacity, i.e. $f = 0.5$. These figures were obtained on a 2GHz AMD EPYC 7662 multi-core server with 100GB of main memory. As it can be seen, the cost of a Simulation phase does not depend on the number of epochs, while the cost of a Prediction phase decreases as the number of epochs increases. As a result, the computational cost of the forecasting procedure grows linearly with the number of epochs.

**Table** *4.1: Maximum elapsed times vs. number of epochs to forecast L2C2 from start to 50% capacity.*

| # epochs | One Simulation phase, minutes. | One Prediction phase, minutes. | Total forecast, days. |
|---|---|---|---|
| 16 | 210 | 38 | 2.8 |
| 32 | 210 | 21 | 5.1 |
| 64 | 210 | 11 | 9.8 |

On the other hand, as it was shown in Figure 4.4, after a certain number of epochs the forecasting error is negligible. Given this trade-off, all the forecasts presented below are made with E = 16 epochs, a good balance between error and cost.

### 4.4.3 Specific situations

In all the experiments performed in this work, the forecasting procedure assumes a uniform distribution of writes among sets. This condition is met in most systems either because the workload is diverse over time and produces an even distribution, or because the NV-LLC incorporates good wear-leveling mechanisms among sets, or both.

However, in some scenarios it may be important to take non-uniformity into account. As an example, we can think of an embedded system that always runs the same applications. Here the distribution of accesses to the cache sets may well be non-uniform, encouraging the design and comparison of mechanisms to even out the wear between sets.

Certainly, the forecasting procedure could also be applied in this context, although the model that approximates the new write rates during the Prediction phase would have to be modified. In particular, the new model could no longer use the average write rate of all frames belonging to sets with a given health state. An alternative could be to obtain the approximation from the distribution of write rates of those frames. We think such specialized forecast is entirely feasible, but it is beyond the scope of this paper.

### 4.4.4 Technological projections of lifetime and performance of NV-LLCs

As we have explained so far, the forward-looking behavior of an NV-LLC can be estimated by applying a forecasting procedure that has three key elements, namely, a statistical model of bitcell write endurance, a detailed simulation model of the NV-LLC organization, and a workload. In principle, a new forecast with a change in any of these three elements requires a feasible, but high computation time.

The results showed in Chapter 2, Section 2.5 were obtained for baseline bitcells with given endurance values modelled with mean $\mu = 10^{11}$ and $cv = 0.2 - 0.3$. To obtain results concerning other bitcell endurance values and/or NV-LLC latencies, of course the whole forecasting procedure can be repeated, creating new *RW maps* and changing the latencies in the simulation model.

However, as long as the NV-LLC latencies are assumed constant, it is possible to take advantage of the properties of the linear transformation of Gaussian distributions to reuse the forecast data and obtain projections for other NVM technologies with a different bitcell write endurance values.

Specifically, if an NV-LLC is built with an improved technology, which offers the same cache latencies, but uses bitcells with $k$ times more endurance ($\mu_i = k \cdot \mu_b,\ \sigma_i = k \cdot \sigma_b$), new capacity and IPC indexes as a function of time can be calculated as follows:

- Cap. improved bitcells ($t$) = Cap. baseline bitcells ($\frac{t}{k}$)
- IPC improved bitcells ($t$) = IPC baseline bitcells ($\frac{t}{k}$)

That is, new indexes with improved bitcells at time ($t$) can be obtained from the forecast made with baseline bitcells at an earlier time ($\frac{t}{k}$); see Appendix A.

Thus, from a few reference forecasts, many technology projections can be obtained. Table 4.2 is an example that focuses on two arbitrary, but interesting, indexes: $T_{90C}$ and $T_{90P}$, calculated from the central column forecasts for $\mu = 10^{11}$ and $cv = 0.2 - 0.3$. $T_{90C}$ and $T_{90P}$ are the elapsed times to reduce the rated capacity and performance, measured in IPC, to 90% of the initial values, respectively. Note that the values of $T_{90C}$ and $T_{90P}$ scale linearly with the value of $\mu$. That is, the value of $T_{90C}$ for $\mu = 10^{12}$ is equal to 10 times the value of $T_{90C}$ for $\mu = 10^{11}$. As it can be seen, $T_{90C}$ always trails $T_{90P}$ and L2C2+6 is the best cache organization.

***Table** 4.2: $T_{90C}$ and $T_{90P}$ for FD+6, L2C2 and L2C2+6, varying cv and $\mu$. m = months, y = years.*

| | | Mean number of writes to fail, $\mu$. | | | | | |
|---|---|---|---|---|---|---|---|
| | | $10^{10}$ | | $10^{11}$ | | $10^{12}$ | |
| $cv$ | Cache | $T_{90C}$ | $T_{90P}$ | $T_{90C}$ | $T_{90P}$ | $T_{90C}$ | $T_{90P}$ |
| 0.2 | FD+6 | 3.7m | 3.9m | 3.1y | 3.2y | 30,7y | 32.2y |
| | L2C2 | 7.1m | 7.2m | 5.9y | 6.0y | 58,9y | 60.1y |
| | L2C2+6 | 7.7m | 7.8m | 6.4y | 6.5y | 63.8y | 64.7y |
| 0.25 | FD+6 | 2.8m | 3.1m | 2.4y | 2.5y | 23.5y | 25.4y |
| | L2C2 | 5.7m | 5.8m | 4.7y | 4.9y | 47.3y | 48.7y |
| | L2C2+6 | 6.4m | 6.5m | 5.3y | 5.4y | 53.1y | 54.2y |
| 0.3 | FD+6 | 2.0m | 2.2m | 1.6y | 1.9y | 16.4y | 18.6y |
| | L2C2 | 4.5m | 4.7m | 3.7y | 3.9y | 37.3y | 39.1y |
| | L2C2+6 | 5.1m | 5.3m | 4.3y | 4.4y | 42.5y | 43.8y |
| | | Projections | | Forecast | | Projections | |

These types of indexes can serve as a basis for signing a Service Level Agreement (SLA) with prospective customers. It is plausible to think that a manufacturer can have a portfolio of NVM qualities and technologies and that a customer can choose the product with the best performance/cost ratio for her/his needs. For example, for a smartphone projected for a daily usage of 6 hours at 100% and with an average product life of 1.8 years [8] several cache organizations and manufacturing variabilities of those shown in the $\mu = 10^{11}$ writes/bitcell columns may fit. These figures could be representative of a technology with moderate write endurance, but comparatively inexpensive.

## 4.5 Additional discussion

### 4.5.1 Forecasting workload behavior in cloud data centers: a seemingly similar problem

Forecasting procedures are relevant today in computer architecture literature. Another problem, similarly addressed, consists of forecasting the demand for resources, e.g. CPU, memory, network and storage, and their power consumption in data centers offering cloud computing services [32, 109].

The objective is to manage in advance the virtual machines (VMs) and/or physical resources needed to elastically adapt supply to demand, complying with the quality of service (QoS) parameters specified in the SLAs made with customers [32, 109]. It is usually assumed that the forecasting procedure receives as input the time series of past events and its outcome will feed a management system capable of automatically commanding resource management in advance. Among other activities, such resource management consists of mapping tasks to VMs and VMs to physical servers. This requires making decisions such as adding or removing VMs, or forcing the live migration of a VM to a different server. It is also necessary to provision the hardware resources of VMs (e.g. number of CPUs, amount of memory and storage required, and communication bandwidth) or decide whether to consolidate several VMs on one server, possibly by physically shutting down part of the servers previously dedicated to those VMs.

Forecasting future demand in cloud data centers is an open problem. It is being approached in the literature from many points of view, both in the workload specification and in the forecasting model itself. In the following, we will review some representative recent work to illustrate this variety of approaches [10, 70, 80, 81, 97].

Regarding the specification of the workload, i.e. how to reproduce the history of resource allocation, usage and release when assigning tasks to VMs, two approaches stand out. The first one consists of describing a synthetic workload, either in a static form [10], or from estimated resource life-cycle probabilities [81]. The second, more widespread, considers time series recorded in real data centers annotated with the relevant events [70, 80, 97].

As for the forecasting model, there are numerous approaches. Most, but not all, are based on machine learning (ML), although no definitive winner is emerging at the moment. For example:

- Bouaouda et al. compare two algorithms taken from the area of operational research to estimate the energy consumed by a cloud data center, namely Ant-Colony Optimization, a population-based metaheuristics, and First Fit Decreasing, an algorithm to solve the Bin Packing problem [10]. The relevant workload events are generated by Cloudsim, a federated cloud data center simulator [13].
- Li investigates how to obtain the maximum benefit, i.e., the best performance/cost ratio, by managing the provision of resources based on the analytical solution of a multi-variable optimization problem [81]. A synthetic workload, defined by probability distributions (task arrival rates, task execution times, waiting times, etc.), is assumed to be executed by multiple computing clusters, consisting of heterogeneous servers of varying speed deployed in a federated cloud environment. Each computer cluster is modeled as an M/M/m queuing system. Two energy cost models are assumed according to the dynamic consumption in idle state. In addition, a benefit/cost model is proposed that considers service revenues, energy expenses and infrastructure amortization charges.
- Finally, we review a selection of recent works based on ML to predict workload behavior [80, 97], including also its energy consumption [70]. All proposals forecast future behavior based on time series (called traces) obtained from real data centers of providers such as Alibaba, Bitbrains or Google. Khan et al. consider several typical ML algorithms such as linear regression, Bayessian Ridge Regression, Automatic

Relevance Determination Regression, elastic nets, and finally, a deep learning (DL) approach, the Gated Recurrent Unit, a particular class of recurrent neural networks (RNNs) that proved to be the best [70]. Leka et al. propose to handle the time series by chaining two neural networks, first a one-dimensional CNN (1D-CNN), which is very suitable for extracting features that relate VMs to each other, and then a Long Short-Term Memory (LSTM) network, another particular class of RNN, to perform the temporal processing of the extracted features and make the forecast [80]; however, the success of the proposal is assessed by comparing it only with CNN or LSTM working separately. Lastly, Patel et al. propose a similar idea in which the 1D-CNN network consists of three parallel dilated 1D layers with different dilation rates CNN (1D-pCNN) to learn CPU load variations at different scales [97]; in addition, the LSTM layer that learns temporal dependencies is fed not only with the patterns recognized by the 1D-pCNN, but also with the original CPU utilization values present in the input time series. Unlike the previous work, Patel et al. compare the forecast errors with a much larger number of alternatives, but only in the area of DL networks.

However, an observation common to the previous referenced work, and to most of the literature on workload and energy forecasting, is the absence of cross-comparisons between complex and simple models [85]. For example, simple statistical methods are rarely used as a baseline for forecasting, making it difficult to quantify the advantage provided by the very expensive methods that rely on complex DL models.

Regarding our purpose of predicting the progressive degradation of the NV-LLC present in a multicore chip of a computing server, we can make several observations. 1) The specification of the problem is very different, in our case there are no time series of degradation events, because NV-LLCs are a pre-industrial product and, if they exist, such traces have not been made public. 2) Related to the above, it is not possible to quantify the goodness of the solution with the typical error metrics that compare reality and predicted value, i.e. the root-mean-square error (RMSE). The validation of our model needed to be done in another way, see Section 4.4.1. 3) The forecasting behavior in data centers is based on reproducing a resource demand that does not follow any known law, while the degradation of NVM bitcells is governed by write reiteration and its Gaussian behavior is well accepted, see Section 1.1.4. 4) The modeled hardware in a data center is assumed to be functional and fault-free, there is still no work that incorporates the detection, diagnosis and repair/replacement life cycle of servers, storage or routers into the forecast; on the contrary, in our case, the main assumption is the existence of a performance-critical component, the NV-LLC, whose capacity, and with it the system performance, will progressively decrease.

Therefore, we can conclude that despite the variety of procedures for forecasting the behavior of cloud data centers, it is not possible to adapt them to our problem, whether they are statistical, operational research or deep learning methods.

## 4.6 Concluding remarks

This chapter comprehensively described a procedure that allows to comprehensively forecast the evolution over time of NV-LLCs whose effective capacity drops due to repeated write operations. To the best of our knowledge, the proposed forecasting procedure is the first in its class. It couples simulation phases in which statistics are gathered from the system with prediction phases in which the bitcells that become faulty are predicted. This methodology has allowed us to compare several NV-LLC organizations in terms of lifetime and performance. It has also allowed us to measure the influence of manufacturing process variability on these results.

Knowledge of how performance evolves through time could be essential for manufacturers to be able to incorporate NVM technologies with the confidence that they can guarantee certain performance for a reasonably appealing time period.

Finally, the new forecast procedure leaves the door open to detailed evaluation of different cache organizations, varying, for example, content management policy between cache levels, replacement policy, or wear-leveling.

*The pawns in the game, they get capped quick.*
*They be out early.*

D'Angelo Barksdale, The Wire.

# 5 HyCSim: A rapid design space exploration tool for emerging hybrid LLCs

*Recent years have seen a rising trend in the exploration of NVMs in the memory subsystem. Particularly in the cache hierarchy, hybrid LLC solutions are proposed to meet the wide-ranging performance and energy requirements of modern days applications. These emerging hybrid solutions need simulation and detailed exploration to fully understand their capabilities before exploiting them. Existing simulation tools are either too slow or incapable of prototyping such systems and optimizing for NVM devices. To this end, we propose* HyCSim, *a trace-driven simulation infrastructure that enables rapid comparison of various hybrid LLC configurations for different optimization objectives. Notably, HyCSim makes it possible to quickly estimate the impact of various hybrid LLC insertion and replacement policies, disabling of a cache region at byte or cache frame granularity for different fault maps. In addition, HyCSim allows to evaluate the impact of various compression schemes on the overall performance (hit and miss rate) and the number of bytes written to both the NVM and SRAM parts of the LLC. Our evaluation on ten multi-program workloads from the SPEC 2006 benchmarks suite shows that HyCSim accelerates the simulation time by* 24×, *compared to the cycle-accurate Gem5 simulator, with high-fidelity.*

## 5.1 Introduction

Off-chip memory accesses continue to be one of the leading performance bottlenecks of multiprocessor systems [74, 84]. This has led to a widespread adoption of LLCs that prevent some requests from private caches from reaching off-chip memories. Increasing the LLC capacity can help reduce the memory wall and meet the larger working set sizes of modern applications. However, traditional SRAM-based LLCs do not seem to scale well to greater sizes due to their higher static power and limited density. Emerging NVMs have been proposed as an alternative to SRAM due to their lower static power and higher density [7, 74, 77, 101, 130, 137]. These technologies, in turn, suffer from limited endurance and costly write operations. Indeed, write operations, in addition to being more time- and energy-consuming than read operations, eventually degrade the bit cells, causing them to lose their storage capacity.

Hybrid cache designs attempt to combine the best of both technologies, i.e., NVMs provide a larger capacity to increase the cache hit rate, and SRAM or DRAM can take up most of the write operations, reducing the dynamic power and increasing the endurance of the NVM device [19, 49, 84]. These hybrid designs offer a huge set of design parameters and design space for exploration. For instance, in typical hybrid LLC designs, write-intensive cache blocks are dynamically mapped to SRAM and read-intensive blocks are mapped to NVMs since they are generally read-friendly devices. However, in hybrid LLCs with compression, the read and write operations feature different tradeoffs due to blocks with different sizes and need more detailed exploration. Similarly, unified (i.e. non-hybrid) cache insertion/replacement policies and compression policies may not be sufficient as the two technologies have different limitations. To explore and evaluate these design alternatives, simulation tools are needed that accurately model different hybrid designs and their content management policies.

Detailed cycle-accurate and full-system simulators such as gem5 and sniper [16, 83] are several orders of magnitude slower than real machines running workloads [11] and are thus impractical to evaluate hundreds of cache configurations. For sensitivity analysis, serial simulations and even minimal modifications to a configuration force researchers to wait hours, and sometimes days, before obtaining the desired result. In addition to this, these tools do not support hybrid cache models and their particular policies. To fill this gap, we propose a flexible trace-driven simulating infrastructure, called HyCSim and whose code is available so that anyone can use it for research purposes [35], that enables rapid exploration of the design space of hybrid LLCs. Our infrastructure makes it possible to quickly assess the interaction between different optimization schemes such as policies for insertion, migration or replacement, mechanisms for data compression, or support to fault management. As starting point, HyCSim requires traces generated from a full-system simulation which can be subsequently used in a series of simulations for the same or different cache models. Our fidelity analysis shows that HyCSim produces simulation outputs with acceptable variations (see Section 5.4.2).

More specifically, our contributions are:

- **Rapid exploration of LLC**: Compared to the cycle-accurate gem5 simulator, HyC-Sim reduces the simulation time by $24x$. If not available already, our tool requires a single run of an application in a cycle-accurate simulator to collect memory traces. HyCSim can then swiftly evaluate a series of cache configurations with high-fidelity.

- **Built-in fault support**: HyCSim is the first simulator to allow disabling memory regions at different granularities in order to explore different content management policies in sets with defective frames or in frames with defective bytes. In this work we call cache frame the array of physical bits that hold a cache block.

- **Compression support**: HyCSim allows designers to explore different block sizes, generated with different compression mechanisms. For now, the compression schemes themselves are not implemented in our tool, so the compression information is exported as part of the generated traces. However, these mechanisms can be implemented in HyCSim in a straightforward way.

- **State-of-the-art policies for hybrid LLCs**: In HyCSim we implemented state-of-the-art policies from the literature. Making these policies available within a single open-source framework will enable future research and contribute to a better reproducibility of research.

## 5.2 Related work

Simulation tools are essential to implement and characterize new architectures and understand their behavior and bottlenecks. Some simulators support modeling an entire system for processor-based systems, i.e., processors and cores, the cache hierarchy, on-chip, and off-chip interconnection frameworks. In contrast, others only focus on a detailed exploration of the memory system or the cache hierarchy. Brais et al. [11] present a survey of the state-of-the-art cache simulators supporting different types (functional, timing), modes (trace-driven, execution-driven), and levels (application-level, full system) of cache simulation. Execution-driven cycle-accurate simulators such as gem5 [83] or sniper [16] better capture microarchitectural features and the impact of various design decisions on the overall system. However, they are considerably slower and therefore impractical to explore the space of hundreds of hybrid cache configurations.

Trace-driven simulation is an attractive alternative and is widely used by the community to find optimal system and cache configurations. For the cache hierarchy, in particular, a number of simulators exist that accurately model the behavior of all/certain cache levels in the hierarchy [11]. For instance, Cachegrind [93], a stable open source project, supports first and last level caches. Dinero IV supports the entire memory hierarchy but primarily reports only hit and miss information [34]. Similarly, CASPER [61] and Pycachesim [50] model the entire cache hierarchy with various standard coherence protocols and standard insertion/replacement policies. Contrary to these multi-level cache simulators, CMP$im [64] and RExCache [113] model unified LLCs with different configurations, but they are only academically published and are closed-source.

All these simulation tools model only unified cache designs. The recent trend in hybrid designs, employing emerging technologies, requires new tools to model their diverse set of properties. For instance, in NVMs, the write operations are not only slow but also harmful and can result in defective bitcells. There are many proposals in the literature to cope with defective bitcells, i.e., providing memory structures with ECCs or redundant (spare) information [110, 133]. In addition, there are mechanisms that allow to deactivate defective regions from the nominal operation, i.e., *frame disabling* (similar to the Intel Cache Safe Technology) [17, 129]. Therefore, the simulation tool should allow marking NVM regions as defective in order to compare different content management policies over a hybrid LLC with degraded capacity.

In addition to the error correction schemes, a number of optimization schemes are proposed to better exploit the performance and lifetime of the hybrid models. For instance, cache insertion and migration policies are presented that aim at reducing the number of write operations in the NVM device [19, 84, 123]. None of the existing open-source simulating infrastructures allow for modeling hybrid caches, and their optimizations, to the best of our knowledge. NVMain [100] and NVmain-ext [69] allow modeling unified NVM-based main memory and LLCs but do not implement any NVM specific policies and optimizations.

## 5.3 HyCSim infrastructure

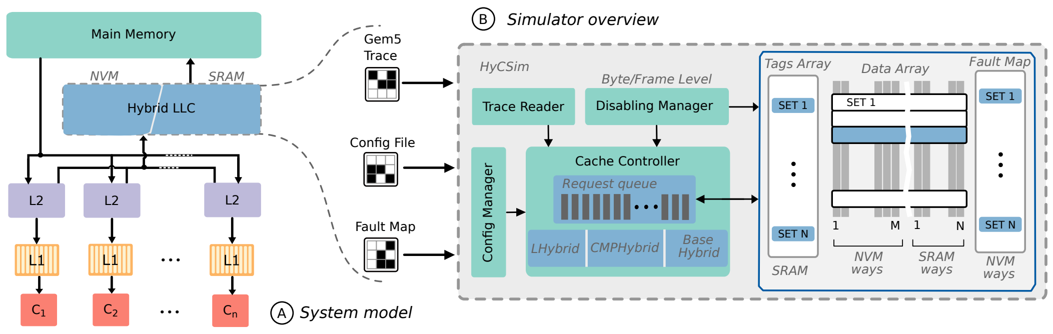


***Figure** 5.1: Ⓐ Overview of the modeled cache hierarchy. Ⓑ Input files, main functions and data structures of HyCSim.*

The HyCSim simulator models the full functionality of a hybrid LLC (SRAM-NVM) in order to evaluate different content management mechanisms and compare them in terms of performance (hit/miss rate) and number of writes to the NVM part. An overview of the infrastructure is depicted in Figure 5.1. There is a cache controller that implements conventional cache functionalities like the address-to-set mapping function, insertion of blocks and the LRU replacement policy. The BaseHybrid (Section 5.3.3), the LHybrid (Section 5.3.3), and the compression-aware models (Section 5.3.3) extend the cache controller functionalities accordingly.

The simulator takes the following files as inputs:

1. The configuration file includes general cache parameters such as size, associativity, and disabling granularity among others.

2. The trace file collects all memory requests coming to the LLC from a profiling run. The traces can be extracted using any tool. We use gem5 for this purpose.

3. The fault map file indicates how many faulty bytes each NVM frame has. This is needed by the fault-aware mechanism to disable at different granularities.

To schedule requests entering the LLC, a first-come, first-served (FCFS) policy is used, i.e., requests in the trace file are served by the cache controller in the order of their arrival. Three data structures are required to carry out the functional simulation, namely: *i*) SRAM tag arrays, for both SRAM and NVM data arrays, *ii*) the data arrays themselves, divided into NVM and SRAM ways, and *iii*) the fault map for NVM ways. In the following sections, we explain the modules of the simulator.

As output, the simulator generates performance statistics such as cache miss and hit rates, number of bytes written per frame, number of LLC bypasses, number of migrations and number of LLC reads and writes. We use the hit rate and the bytes written per NVM frame for performance and lifetime comparison among different models.

### 5.3.1 Cache organization

HyCSim models hybrid set-associative LLC designs consisting of a unified tag array and a hybrid NVM-SRAM data array. It is fully configurable and is not restricted to specific technologies. Technology-specific parameters for the tag and data arrays can also be defined in the configuration file.

The configuration file also defines hybrid design specific parameters, in addition to the standard cache parameters. For instance, the associativity of the NVM and SRAM regions in the data array can be different (e.g., 4-way SRAM and 12-way NVM as in our target system in Table 5.1). Similarly, the two technologies can have different replacement policies, as explained in Section 5.3.3, which can be defined in the configuration file.

### 5.3.2 Disabling manager

Disabling can be done both at byte and frame level. For this, the *Fault Map* that points out defective bytes must be managed accordingly. Disabling with byte granularity results in frames with different capacities. To cope with this, data compression has been proposed in the literature to compress blocks so that they fit in frames with reduced capacity [45]. The cache controller has to know which bytes are and are not defective to perform read and write operations. To this end, the Fault Map consists of a bit per byte that indicates whether the byte is defective or not. We assume that there is a wear-leveling mechanism that is able to evenly distribute the write operations across the alive bytes within a frame [2].

### 5.3.3 Content management policies

Non-inclusive hierarchies have been widely used in the literature, specially in order to avoid redundant data-fills (insertions) that are harmful for NVMs [19]. Such a non-inclusive hierarchy aims at minimizing the number of write operations and thus increase the lifetime of NVMs. Potential LLC block insertions take place when the blocks are evicted from the private levels and the blocks are not present in the LLC. The simulator implements various insertion and data handling policies (see the three boxes under cache controller in Figure 5.1) as explained in the following.

### BaseHybrid

The baseline hybrid LLC model (*BaseHybrid*) assumes *frame disabling*, i.e., the entire frame is disabled as soon as a single byte is faulty. The insertion policy for incoming blocks is based on a global LRU replacement algorithm, which will point to the lowest priority way, either from the SRAM or from the NVM alive ways, as in the reference models used in [19, 84]. That is, a single LRU priority list is used for all ways in both the SRAM and alive NVM ways. Further default parameters can be found in the third row of Table 5.1.

### LHybrid

Hybrid memory structures need special content management policies to take into account various trade-offs. Chen et al. classify all the blocks coming to the LLC into loop blocks, and non-loop blocks [19]. A *loop block* is a clean block that goes back and forth from the LLC to the private levels. Keeping these blocks in the NVM part prevents the NVM ways from suffering from redundant write operations. Therefore, they proposed *LHybrid*, an insertion and migration policy with the objective to avoid write-intensive blocks to be placed in the NVM part in order to reduce the energy consumption [19]. Since LHybrid is a renowned management policies for hybrid cache designs, HyCSim has built-in support for it. Every block has a loop block bit that is set to 1 if the block is clean and has completed one round between the private levels and the LLC. Only the blocks whose loop-block bit is set are placed in the NVM part. Other non-loop blocks are placed in SRAM.

### CMPHybrid

This model supports a compression-aware insertion policy. For compression support, the simulator needs to receive either the compressed size of each block or the block's bitstream to calculate the compressed size. In the current version of HyCSim, we assume that the block compression information is exported with every LLC write request in the trace file. However, if different compression schemes are needed to be explored, they can also be implemented at the HyCSim end. For this, the block's bitstream should be included in the trace file.

CMPHybrid implements byte disabling and the following insertion policy. Blocks are inserted either in SRAM or in NVM ways according to a global LRU replacement algorithm, similar to the BaseHybrid. However, unlike BaseHybrid, the disabling takes place at the byte granularity, and CMPHybrid intelligently exploits the fault map information in the replacement algorithm. NVM frames having a capacity lower than the incoming compressed block sizes are not considered as replacement candidates.

## 5.4 Validation

In this section we compare HyCSim with a reference simulator, gem5 extended with the Ruby memory subsystem, obtaining the simulation time and metrics of interest. For each selected configuration the reference simulator produces a trace of LLC accesses that will then be used to feed HyCSim, simulating, in turn, the same or another configuration.

***Table*** *5.1: Baseline system configuration*

| | |
|---|---|
| Cores | 4, out-of-order, 3.5 GHz. |
| L1<br>L2 | Private, 32 KB D, 32 KB I, 4 ways, LRU.<br>Private, inclusive, 128 KB D, 128 KB I, 16 ways, LRU. |
| Hybrid LLC | Shared, non-inclusive, 8MB, 4 SRAM ways, 12 NVM ways, LRU. 64B block |
| Main Memory | 1 memory controller, DDR4. 1 channel, 8GB/channel (1200 MHz) |

***Table*** *5.2: Configurations tested*

| Model | LLC size | % of frames (F) or bytes (B) alive in the NVM part | Acronym |
|---|---|---|---|
| BaseHybrid | 8MB | 100%F | BH |
| | | 75%F | BH_75% |
| | 16MB | 100%F | BH_16MB |
| LHybrid | 8MB | 100%F | LH |
| | | 75%F | LH_75% |
| | 16MB | 100%F | LH_16MB |
| CMPHybrid | 8MB | 100%B | CH |
| | | 75%B | CH_75% |
| | 16MB | 100%B | CH_16MB |

The multiprocessor system modeled consists of 4 cores, each with two private cache levels (L1 and L2) and an LLC, see Table 5.1. The coherence protocol is a directory-based MOESI and the on-chip interconnection network, which is a crossbar, connects the private levels, the LLC and the directory.

The workload consists of ten mixes randomly created from SPEC CPU 2006 benchmarks [56]. The applications with low LLC activity have not been included to form the mixes [92].

The configurations selected appear in Table 5.2. They correspond to the three models explained in Section 5.3, BaseHybrid, LHybrid and CMPHybrid. For each model with an LLC of 8MB, a reduction in the capacity of the NVM part is considered, resulting in 75% of available frames (BaseHybrid and LHybrid) or bytes (CMPHybrid). Furthermore each model is tested with a 16 MB LLC with all the NVM capacity available.

### 5.4.1 Simulation time analysis

Figure 5.2 shows the increase in simulation speed achieved by HyCSim over gem5. Each bar represents the simulation speedup when running a specific mix of the workload with all the cache configurations in Table 5.2.

The tool accelerates the simulation by 24×, on average (geometric mean). In addition, we can see that between Mix4 and Mix8 there is more than 4× difference in the speedup. This is because different mixes have different computation and memory requirements. HyCSim

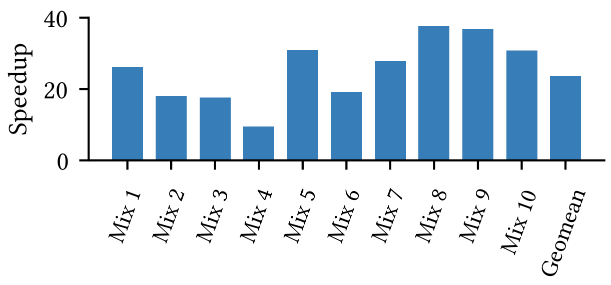
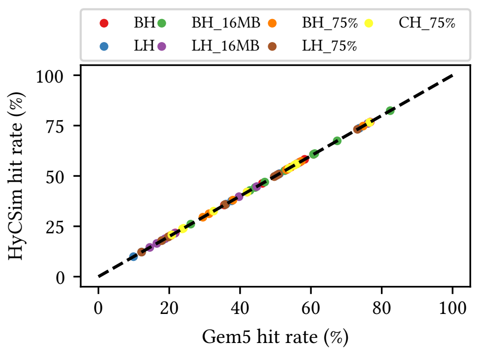
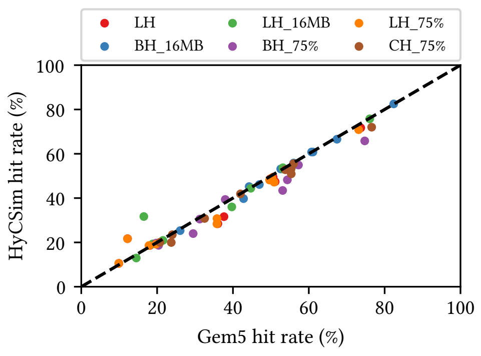


***Figure** 5.2: Simulation speedup compared to gem5.*

only simulates the LLC while gem5 simulates the whole system. The simulation of mixes with more LLC activity is not accelerated as much as simulations of computation-intensive mixes.



(a) Exact traces



(b) Different traces

***Figure** 5.3: Fidelity study. Each dot represents the hit rate provided by HyCSim and its corresponding hit rate in gem5.*

### 5.4.2 Fidelity analysis

In order to validate our HyCSim tool, we compare the simulation output of HyCSim with that of the full-system simulation of gem5. Each cache configuration of Table 5.2 has been simulated in the reference simulator, gem5, to obtain the metrics of interest and the trace of LLC accesses. With the obtained traces, two experiments have been performed to check the fidelity of the results obtained with HyCSim. First, each configuration was simulated in HyCSim using the trace generated by gem5 for the same cache configuration. The comparison with the results obtained by gem5 shows the error committed by inaccuracies in the implementation of each configuration. Secondly, each configuration has been simulated in HyCSim using always the trace generated by the base configuration (BH configuration in Table 5.2). The comparison with the results obtained by gem5 shows the inaccuracy introduced by the use of a trace generated with a configuration different from the simulated one.

The comparison in terms of LLC hit rate for the first experiment is shown in Figure 5.3a. The X-axis represents the hit rate obtained by gem5 and the Y-axis represents the hit rate obtained by HyCSim. Each point shows the values obtained for a configuration with the trace of an application mix. If the hit rate provided by HyCSim matches that of gem5, the dot is plotted over the black dashed line. As we can see, almost all the dots are over the black line, meaning that HyCSim is able to capture the full LLC functionality accurately.

However, feeding each cache configuration in HyCSim with a trace generated using the same cache configuration in gem5 is not feasible. It would require modeling and simulating each configuration in the full-system simulator. This turns out impractical during design space exploration and it is precisely what we want to avoid with the new simulator. Therefore, traces are usually extracted using a reference cache configuration in gem5 and are used to simulate many other different configurations in HyCSim. Figure 5.3b depicts the hit rate provided by HyCSim when it is fed with traces extracted with a different LLC model in a cycle accurate simulation. We can see that most of the dots remain at or near the black line, which means that the tool is accurate enough to provide statistics even when it is fed with different traces, correctly guiding design decisions. In terms of fidelity, the data features a Spearman's correlation factor of $\rho = 0.968$.

### 5.4.3 Proof of concept

In this section, we present a proof of concept of our simulator. Specifically, we use HyCSim to obtain the number of bytes written to the NVM frames of a hybrid LLC under the three content management policies presented in Section 5.3.3. To complete the analysis, we model a configuration that uses the LHybrid insertion and migration policy and also uses compression to reduce the number of bytes written in the NVM part. All models use an 8 MB cache with all NVM frames available.

Figure 5.4 shows the number of bytes written to NVM frames using each management policy normalized (in logarithmic scale) to the base configuration (BH in Table 5.2). Data are shown for each workload mix and the average for all of them. LHybrid reduces writes to NVM frames by 95.7% on average, with a range between 87.2% and 99.7%. Data compression manages to reduce writes by 41.3% over the base system and 44.7% over LHybrid on average. This preliminary analysis suggests that data compression can be an effective technique to reduce the number of writes on NVM frames even in a cache that already incorporates other techniques with the same objective as in the case of LHybrid.

## 5.5 Concluding remarks

This chapter introduced HyCSim, a tool for rapid design space exploration of hybrid LLCs. Compared to the cycle-accurate full-system simulator gem5, HyCSim reduces the simulation time by 24× (geometric mean). It implements state-of-the-art LLCs content management policies and a disabling manager that allows disabling defective NVM regions at different granularities. HyCSim is trace-driven and uses a generic trace format including cache request type, cache block address and compression information, and the generating core ID. The sources of the simulator along with configuration files are available open source [35].

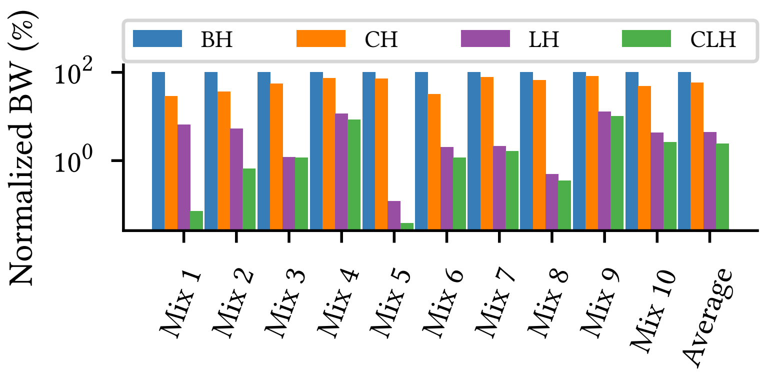


***Figure** 5.4: Bytes written (BW) to NVM frames normalized to BaseHybrid.*

With ten random mixes of the SPEC 2006 benchmarks, we demonstrate that for the same cache model, HyCSim generates the same output statistics as gem5. For different cache models, our fidelity analysis shows acceptable variations in the outputs of the simulations. Note that, similar to any other trace-driven tool, this may not be the case if entirely different cache models are simulated. For instance, a hybrid LLC model placed in a different inclusive/exclusive hierarchy should need traces extracted from such a hierarchy. HyCSim can be used to investigate different cache designs by changing design parameters. As an example and proof of concept, we evaluate the content management policies in terms of number of bytes written to the NVM part of a hybrid cache. More important, HyCSim helped us with the design space exploration of the insertion policies proposed in Chapter 3. More specifically, the hit rate and bytes written figures of Section 3.4 are extracted from this tool.

## Part III

# Compute-in-memory using NVMs

*We ain't gotta dream no more.*

Stringer Bell, The Wire.

# 6 Extending MNEMOSENE, an NVM-based compute-in-memory general purpose architecture

*NVMs have shown their potential within the compute-in-memory (CiM) paradigm due to their memristive properties. This chapter optimizes the MNEMOSENE architecture, a CiM tile design integrating computation and storage for increased efficiency. We identify and address bottlenecks in the Row Data (RD) buffer that cause losses in performance. Our proposed approach includes mitigating these buffering bottlenecks and extending MNEMOSENE's single-tile design to a multi-tile configuration for improved parallel processing. The proposal is validated through comprehensive analyses exploring the mapping of diverse neural networks evaluated on CiM crossbar arrays based on several NVM technologies. These proposed enhancements lead up to 55% reduction in execution time compared to the original single-tile architecture for any general matrix multiplication (GEMM) operation. Our evaluation shows that while ReRAM and PCM offer notable energy advantages, their integration with scaled CMOS is limited, which leads to voltage-gate assisted SOT (VGSOT)-MRAM emerging as a promising alternative, due to its good balance between energy efficiency and superior integration capabilities. The VGSOT-MRAM crossbar arrays provide 12×, 49×, and 346× more energy efficiency than PCM, ReRAM, and STT-MRAM ones, respectively. It translates, on average for the considered workload, in 1.5×, 3×, and 14.5× better energy efficiency of the entire system.*

## 6.1 Introduction

In the pursuit of improving the energy efficiency and computational prowess of future generations of computers, the focus has steadily shifted towards compute-in-memory paradigms. Contrasting the traditional von Neumann architecture that segregates computing and memory units, in-memory computing seamlessly integrates these units to yield significant energy savings. A vast body of research has already been undertaken on the design of memory arrays and their peripheral circuitry, with various emerging NVM technologies, such as ReRAM, PCM, STT-MRAM or VGSOT-MRAM [6, 68, 114, 138].

One of the distinctive architectures in this domain is the compute-in-memory-periphery (CiM-P) tile [134, 135], where storage and computation are performed in the analog memory array, with the resultant data delivered in the digital periphery. This approach not only promotes energy efficiency but also presents a more streamlined operational model.

The MNEMOSENE CiM architecture is an epitome of this approach, featuring a NVM array and peripheral circuitry with a defined ISA for bridging higher-level programming languages to the underlying circuit designs [134, 135]. However, as with all leading-edge architectures, their first implementations may present large margins for improvement. This chapter focuses on the shortcomings of the current design, particularly in relation to the concept of a single-tile implementation and the limitations of memory transactions with the internal buffers.

This chapter's key contribution is threefold: We first propose to revisit the current internal buffering in the MNEMOSENE tile architecture to address the bottleneck caused by expensive system memory requests. To this end, we suggest implementing a double input buffer to decouple the data load to the tile from the analog computation, alleviating such a bottleneck. This optimization aims at reducing performance overheads and enhance energy efficiency.

Secondly, we extend the MNEMOSENE single-tile design towards a multi-tile microarchitecture. To this end, we suggest using a shared scratchpad memory and an efficient interconnection network. We propose a co-design procedure that optimally and simultaneously sizes the scratchpad memory and the interconnection network, further mitigating unnecessary delays. This co-design ensures efficient data retrieval, and enables seamless communication across multiple tiles, thus increasing the computational capabilities of the overall ensemble.

Finally, to validate the proposal, we perform a comprehensive analysis of both small and large CNNs, ranging from TinyML and AnalogNet to more extensive networks like Nasnet and Resnet. The performance of these CNNs under the enhanced MNEMOSENE architecture provide valuable insights into the potential of our proposed modifications. Finally, we perform an extensive evaluation using in-house MRAM technologies, such as STT-MRAM and VGSOT-MRAM, and state-of-the-art PCM and ReRAM. This evaluation offers a broader perspective on the implications of our proposed enhancements. The comprehensive analysis underscores the effectiveness of faster, low-power NVM technologies such as VGSOT-MRAM for energy-efficient CiM computations. The suitability of VGSOT-MRAM stems from its ability to utilize reduced currents during inference time, a characteristic attributed to its higher resistances.

The subsequent sections of this chapter delve into the specifics of the proposed extensions to the MNEMOSENE architecture, followed by the analysis of neural network performance and the comprehensive evaluation of the improved architecture under different NVM technologies.

## 6.2 Background

In the quest for more energy efficient and high performance computing, the CiM paradigm using NVMs has emerged as a promising prospect. This paradigm exploits the memristive properties inherent in NVM technologies. These memristive properties provide NVMs with the ability to program the resistance of the cells, that is, the electrical resistance of NVM cells changes in response to applied voltage or current in order to code numerical values.

### 6.2.1 Analog computing-in-memory using NVMs

The NVM crossbar array is the backbone of the analog CiM paradigm. NVM cells are analogously conformed in an array fashion as in traditional memory arrays. Integer values are coded as resistance values in one or more NVM cells within the crossbar array. By properly sensing the cells, it is possible to perform from Boolean operations like bitwise AND, OR, XOR, etc. to simple arithmetic operations as additions or dot products.

The dot product is the intrinsic operation in a GEMM, which is one of the pivotal tasks in ML workloads such as CNNs. Figure 6.1 shows a simple example of an analog dot product operation. Two NVM cells (or *resistors*) are placed in two different rows and share the same column. Two integer values are coded in the magnitude of the resistance of the two resistors, $R1$ and $R2$. The other two integers taking part in the dot product are coded in the form of voltages, $V1$ and $V2$. These voltages are fed into the crossbar array through the Digital-to-analog converters (DACs). The magnitude of the current that each row is feeding into the column is proportional to the product of the integers coded as $R1$ and $V1$ for the first row, and as $R2$ and $V2$ for the second row. The result of the dot product is proportional to the sum of the currents fed by both rows, $I1 + I2$, which will be sensed out and decoded by an analog-to-digital converter (ADC) at the bottom of the column.

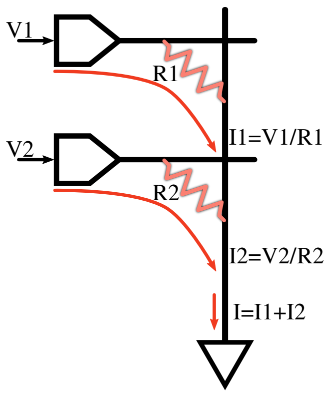
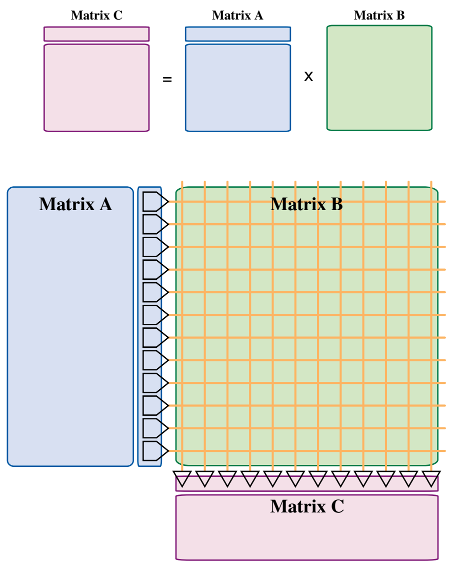
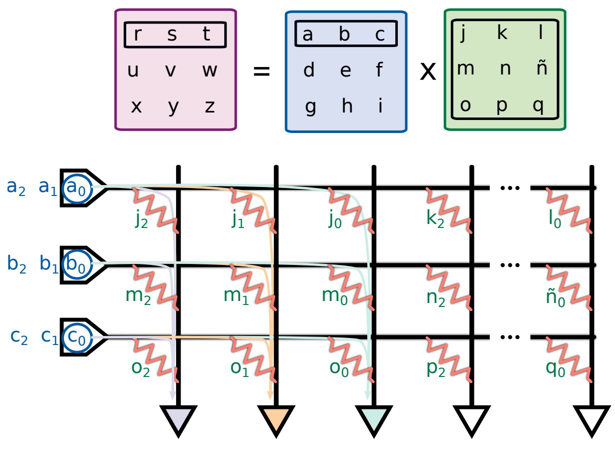


***Figure** 6.1: Analog dot product operation scheme.*

Next we explain a more complex example, Figure 6.2a shows the mapping of a GEMM operation in such a crossbar array. The multiplication of *Matrix A* and *Matrix B* is performed by first coding *Matrix B* as resistance values, which are written in the crossbar array. Then, *Matrix A* is transposed and fed into the crossbar array as voltages through the DACs row by row. Finally, the result, *Matrix C*, is collected through the ADCs in a massively parallel fashion row by row.



(a) Mapping a GEMM



$$
\begin{aligned}
r = {} & 2^0(2^2(a_0j_2+b_0m_2+c_0o_2) + 2(a_0j_1+b_0m_1+c_0o_1) + (a_0j_0+b_0m_0+c_0o_0)) + \\
& 2^1(2^2(a_1j_2+b_1m_2+c_1o_2) + 2(a_1j_1+b_1m_1+c_1o_1) + (a_1j_0+b_1m_0+c_1o_0)) + \\
& 2^2(2^2(a_2j_2+b_2m_2+c_2o_2) + 2(a_2j_1+b_2m_1+c_2o_1) + (a_2j_0+b_2m_0+c_2o_0)) +
\end{aligned}
$$

(b) 3-bit datatype size GEMM example

***Figure** 6.2: Mapping of a GEMM to a NVM crossbar array (a). Example of a 3-bit datatype size* $3 \times 3$ *matrix-matrix multiplication in the crossbar (b).*

To be more specific, we now assume $3 \times 3$ matrices whose elements are 3-bit unsigned integers. Figure 6.2b details the computation of the first element of *Matrix C*, *r* in this GEMM example. Let us assume that, as in the MNEMOSENE framework, each NVM cell codes one bit, either in high or low resistance state (HRS/LRS), and *Matrix B* is already written in the crossbar. Elements *j*, *m*, and *o* are placed in subsequent rows while their corresponding bits, 2, 1, and 0 occupy adjacent columns. The elements of *Matrix A*, *a*, *b*, and *c* are shifted into the crossbar through the voltage drivers bit by bit. The result of every partial dot product is sampled out through the ADCs and gathered together by a digital periphery. This periphery performs a light post-processing to take into account the weight of each bit in the product. The partial dot products are highlighted with the corresponding colour to the one of the column that has generated them, f.i., orange dot products are generated by the second column.

Note that in parallel with the computation of *r*, the computation of *s* and *t* is also taking place thanks to the storage of the elements (*k*, *n*, *p*, *l*, *ñ*, *q*) in the rest of the crossbar array.

## 6.3 Enhancing the MNEMOSENE Tile Architecture

Many studies in CiM have focused on developing standalone accelerators tailored to address specific computational tasks. GEMMs are the dominant operations in today's Machine

Learning workloads. From standard CNN to large transformers networks, the most common layers are accelerated by its conversion to GEMM operations [138, 140].

However, the number of CiM architectures designed to seamlessly integrate with general-purpose multiprocessor systems remains relatively limited. One notable project that exemplifies this integration is MNEMOSENE. In their work, Zahedi et al. present a programmable single-tile architecture alongside an ISA and a compiler that aims to establish a standardized simulation framework for CiM designs [134, 135]. This framework facilitates the creation of a flexible interface between the CiM tile and the broader system, enabling effortless interaction and integration within a general-purpose multiprocessor environment. The defined ISA lies at the core of this integration and aims to enhance the flexibility and generality of the hardware by shifting the complexity towards the compiler.

Note that in this chapter of the thesis the word "architecture" will be used repeatedly to denote what has been called "microarchitecture" previously in this dissertation. This is due to the collaboration with researchers close to MNEMOSENE's pioneering work, and therefore to the adoption of their terminology.

### 6.3.1 Original single-tile architecture

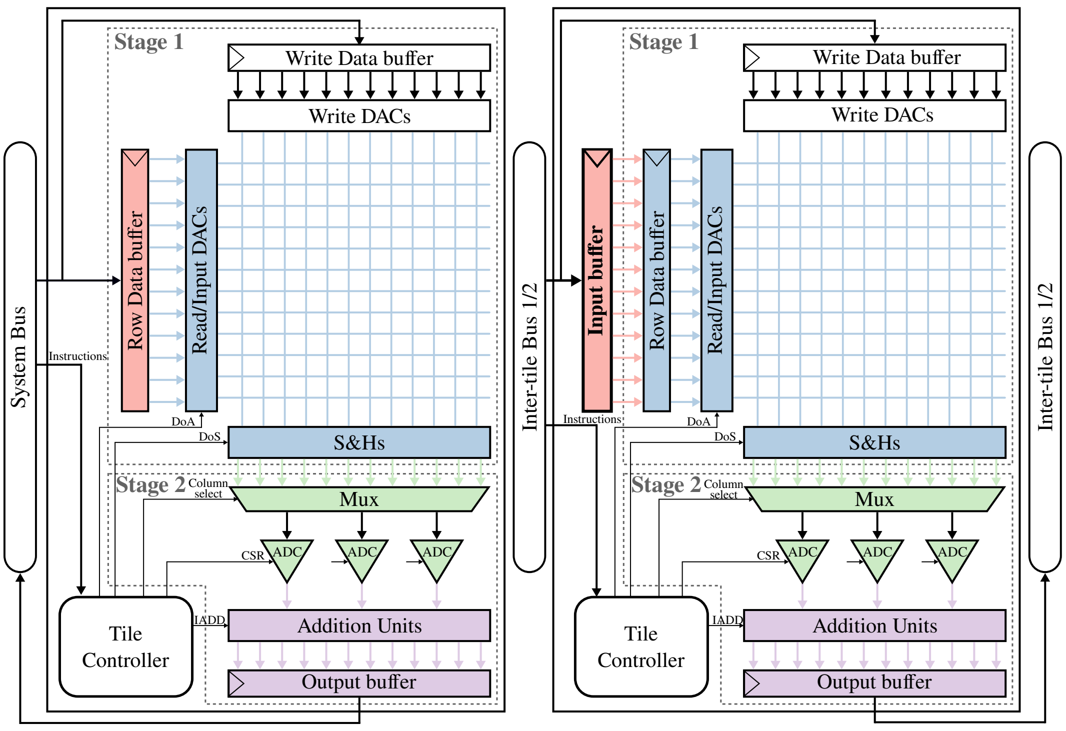
We now show the interactions between the main components of a single CiM-tile architecture. More details on how the different instructions work and what their circuit specifications are can be found in [134, 135].

Figure 6.3a illustrates the original MNEMOSENE single-tile architecture able to run a GEMM. One matrix is stored in the memristive crossbar array through the *Write Data* (WD) buffer while the other one is transferred row by row to the *Row Data* (RD) buffer before the analog computation can take place.

A row computation is pipelined in two stages, Stage 1 on top and Stage 2 below. Aside from these stages, the *tile controller* plays a vital role in receiving nano-instructions, decoding them into various control signals, and efficiently scheduling their execution across each stage. In Stage 1, the RD buffer serves as the recipient for the data elements, which act as inputs for each crossbar row. Then, the incoming elements are shifted one bit at a time into the DACs to perform the analog computation in the memristive crossbar array. The crossbar outputs are then collected from each column by the *Sample-and-Hold (S&Hs)* units. In Stage 2, multiple columns share a single ADC. To handle this scenario, the partial results from each column are sampled by the ADC in a time-multiplexed manner. Following this, the *addition units* combine these partial results to generate the final outcomes, which are then stored in the *output buffer*.

### 6.3.2 Overcoming the RD buffer limitation

Unfortunately, bringing the data to the RD buffer imposes significant overhead in the original single-tile architecture. The tile is required to wait for the system to refill the RD buffer for each subsequent matrix row since the RD buffer is written byte to byte. Consequently, the RD buffer may inadvertently cause delays in standard workloads, thereby hindering the overall efficacy of the compute-in-memory operations.



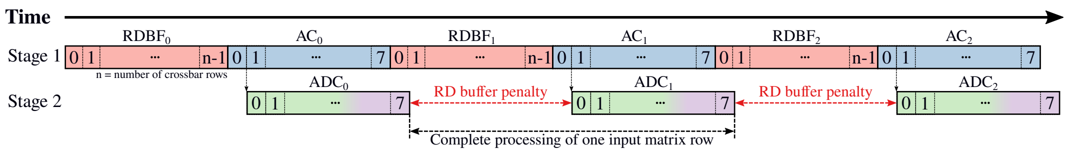
(a) Original single-tile architecture (b) Proposed single-tile architecture

***Figure** 6.3: CiM intra-tile architecture. (a) MNEMOSENE original architecture and (b) double-buffer proposal.*

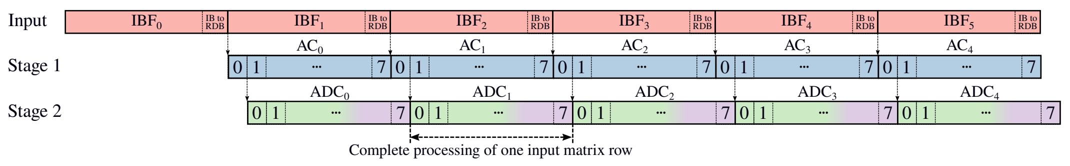
Figure 6.4a illustrates the timing diagram of computing the first row of 8-bit elements GEMM. The timing depicts the breakdown of the execution time among the different components. In the original configuration, the RD buffer has one entry per crossbar row ($n$ rows in the figure) and every entry is 8 bits wide.

Filling the RD buffer consumes one cycle per buffer entry, while the full execution of the 8-bit element requires a delay that is dominated by the ADC conversion. On a permanent basis, transferring the following row to the RD buffer imposes a significant latency overhead that could be overlapped with the analog computation (AC) of the previous row.

Therefore, the goal is to hide the RD buffer latency penalty by overlapping it with the analog computation. To this end, we propose to add an input buffer (IB) of the same size of the RD buffer, see Figure 6.3b, to decouple the data load to the tile from Stage 1. Thus, while the analog computation of the first row is taking place, the second row is being fetched into the IB in the background, see $AC_0$ and $IBF_1$ in Figure 6.4b. While the analog computation of the current row, $AC_i$, is finishing, the next row, already loaded in the IB, is copied to the RD buffer. Assuming the IB can be completely filled during the analog computation of the previous element, the tile pipeline execution is now dominated by Stage 2. In Section 6.4.2 we will further investigate how the rest of the multi-tile components are sized and parameterized so that the IB is seamlessly integrated and can be completely filled in the background to avoid any additional overheads.



(a) Original MNEMOSENE single-tile architecture.



(b) Decoupling the data load to the tile from the analog computation.

***Figure** 6.4: Timing diagram of multiplying the first matrix row in both (a) the original single-tile and (b) the proposed optimization. Colors match the components of Figure 6.3. RDBF stands for RD buffer fill, AC for analog computation, ADC for ADC conversion and addition units, and IBF for IB fill. The penalty imposed by the RD buffer is highlighted in red in (a).*

Our experiments showed that the double input buffer approach leads up to 55% reduction in execution time for a single GEMM compared to the original MNEMOSENE tile architecture.

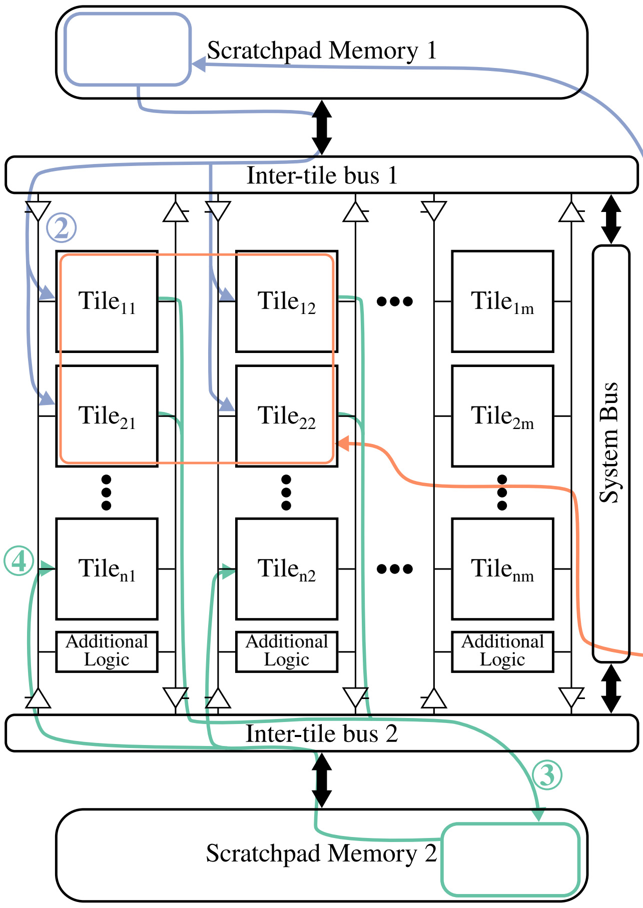
## 6.4 Enabling a Multi-Tile Architecture

### 6.4.1 Multi-tile architecture

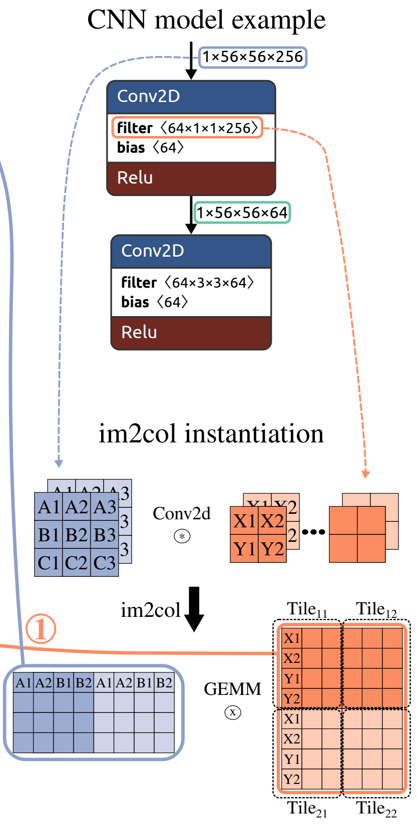
A multi-tile architecture is essential in CiM operations due to inherent scalability and performance constraints associated with single-tile configurations. As computational tasks grow in complexity, the capacity of a single tile to effectively address these requirements diminishes. Transitioning to a multi-tile structure provides the system with the capability to distribute computational tasks across multiple tiles, facilitating parallel processing and augmenting overall performance.

Figure 6.5a depicts the proposed multi-tile architecture. It consists of numerous tiles, logically placed in a two-dimensional fashion, two inter-tile buses and two scratchpad memories. Each bus is connected to a scratchpad memory, allowing decoupling the read and write flows of the tiles. The scratchpad memory serves both as intermediary with the outer system and to enable tile-to-tile communication.

Figure 6.5b shows how a CNN model is mapped into such a multi-tile architecture. The convolutions, which are the pivotal computational task in CNNs, are turned into GEMM operations by means of the image to column (im2col) transformation [140]. The ever-growing dimensions of these matrices hinder them to fit into a single tile. Weight matrices are thereby broken down into chunks and distributed across multiple rows and/or columns of tiles (1). Input matrices are also split up and forwarded to the tiles matching the corresponding weight chunks. While the analog computation takes place, a consistent data stream flows from one scratchpad memory to the IB of the tiles (2), and reciprocally, from the tile output buffer back to the other scratchpad memory (3). In this way, the partial results from multiple tiles are aggregated in the *additional logic* conforming the complete



(a) Proposed multi-tile architecture



(b) Mapping a CNN model to a multi-tile architecture

***Figure** 6.5: CiM multi-tile architecture. (a) Proposed multi-tile organization and (b) mapping a CNN model to the multi-tile architecture. We highlight how after a CNN is transformed to be computed as a GEMM using* im2col, *the different filters in a layer are distributed into a group of tiles. Independent layers access separated areas on the two scratchpad memories, pipelining the read/writes from/to the buffered data.*

result of a GEMM. It is essential to note that the output matrix of one layer may become the input matrix of the subsequent layer 4. Hence, certain layers read from the top-positioned scratchpad and write to the one below, while the adjacent layers operate vice versa.

### 6.4.2 Properly sizing scratchpads and interconnections

To overcome the limitations outlined in Section 6.3.2, it is crucial to optimally size the scratchpad memory and the buses, preventing additional overheads during the population of the tiles' IBs. The architectural components delineated in the previous sections are predominantly dependent on the workload. However, the flexibility of the MNEMOSENE architecture enables comprehensive parameterization, allowing for optimal customization for the task at hand. Experiments were conducted to size the proposed components appropriately, validating the design's feasibility.

To accurately size the scratchpad memory, focus is placed on identifying the CNN layer with the largest memory footprint. This evaluation encompasses not only the dimensions of the input and output matrices, but also takes into account the input data of concurrent residual connections within the CNN. On the other hand, to avoid additional communication overheads between the scratchpad and the tiles, it is essential to ensure that the inter-tile buses can transmit any matrix row from the scratchpad to the tiles within the time frame in which the analog computation ($AC_i$) of the previous row takes place, as depicted in Figure 6.4b. Therefore, the bus width must be properly sized to accommodate the transmission of the matrix row of maximum size within this time frame; note that excessive $IBF_i$ times in Figure 6.4b would lead to undesired delays. Section 6.5.1 shows, for the evaluated workloads, the optimal sizing values we obtained for both scratchpad memory and buses.

## 6.5 Evaluation

### 6.5.1 Experimental Setup

Experiments utilized the MNEMOSENE simulator [134, 135] with a 256x256 analog crossbar array per tile, and 8-bit datatype size. All NVM technologies, ReRAM [53], PCM [76], STT-MRAM [47], and VGSOT-MRAM, are configured in dual-state memory cells; their most important figures are provided in Table 6.1. In the present experiments, the deployment of a weight involves one column per weight bit. Our in-house STT-MRAM devices are accompanied by a projected VGSOT-MRAM device simulation that, making use of a higher voltage-controlled magnetic anisotropy (VCMA) coefficient, relaxes the critical write current and enables 4-pillar bitcells. DACs, S&Hs, and ADCs specifications, as well as a 1GHz operating frequency, matched the original MNEMOSENE papers [134, 135]. Input and RD buffers were modeled with Nandgate 15nm technology [86].

***Table** 6.1: NVM technologies specification.*

| | ReRAM | PCM | STT-MRAM | VGSOT-MRAM |
|---|---|---|---|---|
| Low Resistance State | 5 kΩ | 20 kΩ | 6.2 kΩ | 824.1 kΩ |
| High Resistance State | 1 MΩ | 10 MΩ | 15 kΩ | 2.1 MΩ |
| Memory Read Voltage | 0.2 V | 0.2 V | 0.5 V | 0.55 V |
| Memory Read Time | 10 ns | 10 ns | 10 ns | 3 ns |

The workload comprised two small AnalogNets CNN models (keyword spotting (KWS) and visual wake words (VWW)) [138], and two Tensorflow CNN models (Nasnet [141], Resnet50 [54]). Notably, Nasnet presented the layer with the largest memory footprint, reaching 7.3 MB. To accommodate memory requirements, 2 scratchpad memories of 4 MB each were designed, consisting of eight 512 KB banks each, modeled using an in-house STT-MRAM data memory featuring 22nm technology node, considering latency and energy for comprehensive CNN inference evaluation. For efficient data transfer between tiles and the scratchpad, a 48-byte bus width sufficed.

### 6.5.2 Experimental Results

Tables 6.2 and 6.3 summarizes the results after mapping one inference of the different CNN models to our multi-tile architecture. Looking from a workload perspective, and regarding the memory footprint of the models, the small AnalogNet models use at most 405 KB of the scratchpad memory, 5% of its total size. Larger models thereby have larger memory requirements, with 2.8 and 7.3 MB for Resnet50 and Nasnet, respectively. One complete inference is performed and the latency and energy results are reported. The tile execution is dominated by Stage 2, see Figure 6.4b, so all implementations of the same CNN, regardless of the NVM technology, report the same expected latency. While small models complete one inference within the range of 1 ms, the larger ones take up to 39.5 ms to finish it.

***Table** 6.2: Analyzed CNN models, required number of tiles, computational utilization, and memory footprint.*

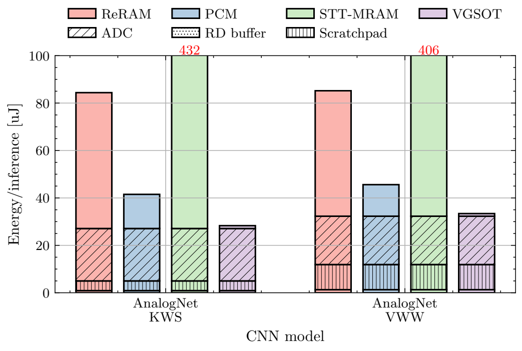
| CNN model | Model size (#Parameters) | Required tiles | Computational utilization (%) | Maximum footprint |
|---|---|---|---|---|
| KWS | Small | 46 (7×7) | 79.8 | 136.5 kB |
| VWW | Small | 75 (9×9) | 57.6 | 405.0 kB |
| Nasnet | Medium (5.3M) | 6364 (80×80) | **8.3** | 7.3 MB |
| Resnet50 | Large (25.6M) | 2966 (55×55) | **97.6** | 2.8 MB |

***Table** 6.3: Latency and energy per inference for the analyzed CNN models.*

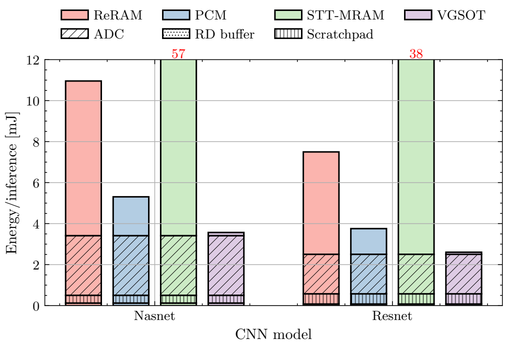
| CNN model | Latency of a single inference | Total energy of a single inference | | | |
|---|---|---|---|---|---|
| | | ReRAM | PCM | STT-MRAM | VGSOT-MRAM |
| KWS | 185.8 $\mu$s | 84.3 $\mu$J | 41.4 $\mu$J | 431.5 $\mu$J | 28.2 $\mu$J |
| VWW | 1.3 ms | 85.3 $\mu$J | 45.6 $\mu$J | 406.1 $\mu$J | 33.4 $\mu$J |
| Nasnet | 39.5 ms | 11.0 mJ | 5.3 mJ | 56.7 mJ | 3.6 mJ |
| Resnet50 | 10.9 ms | 7.5 mJ | 3.8 mJ | 37.8 mJ | 2.6 mJ |

Regarding energy, and due to the different conductances of each NVM technology, the crossbar arrays report varying energy consumption figures for the same CNN model. In particular, for our in-house NVM technologies, STT-MRAM provides ~7× and ~28× higher energy consumption than ReRAM and PCM, respectively. Despite these higher power consumption, the great advantage of STT-MRAM lies in its high integration capabilities with scaled nodes [47]. As a successful tradeoff, VGSOT-MRAM combining reduced area and lower energy requirements, becomes the most promising technology compared to other NVMs. Based on our in-house device projections, the VGSOT-MRAM crossbar arrays are 12×, 49×, and 346× more energy efficient than the PCM, ReRAM, and STT-MRAM ones, respectively. It translates, on average for the considered CNN models, in 1.5×, 3×, and 14.5× better energy efficiency of the entire system.

Figure 6.6 shows the energy per inference breakdown, for the different CNN models, separating the contribution of the NVM crossbar array, the ADC, the RD buffer and the scratchpad memory. As can be seen, the energy numbers differ significantly between CNN models. For instance, for PCM and ReRAM, smaller analog CiM specific CNN models operate in the range of 40 to 400 $\mu$J, while standard larger models (NasNet and Resnet50) consume 2 to 60 mJ per inference. Our research highlights that AnalogNet workloads, optimized for analog CiM accelerators, demonstrated a better utilization of tile resources versus peripheral areas and buffers. Therefore, we can highlight how optimizing its deployment on analog hardware via Network-Architecture-Search (NAS) not only improved neural network accuracy [138], but also significantly boosted energy efficiency, underscoring the value of NAS for these specific accelerators.



(a) Small CNN models



(b) Large CNN models

***Figure** 6.6: Energy/inference (in mJ) breakdown of the different architectural components for the larger CNN models, evaluating distinct NVM technologies.*

The varying energy efficiency across different technologies reveals a significant contrast in the energy consumption of periphery and scratchpad components. On the one hand, the energy consumed by these assisting components is minimal compared to the overall energy for some technologies, see ADC + I/O Buffers + Scratchpad energy in Figure 6.6. For instance, for STT-MRAM, the energy of the assisting components accounts for at most 7.9% of the overall one for VWW. On the other hand, for VGSOT-MRAM, these components emerge as the primary energy factor, up to 96.8% of the overall one for VWW. Further improving the energy efficiency of such low power technologies will involve special efforts to improve the peripheral components as well.

### Scratchpad STT-MRAM vs. SRAM

In order to assess the energy efficiency of the scratchpad memory built using STT-MRAM technology, the proposed STT-MRAM scratchpad is compared against an analogous one, implemented with SRAM technology. For this, the 22 nm SRAM memory model from CACTI [90] is used. In addition, it is verified that both scratchpad memories meet the requirements of the multi tile architecture by checking that undesired delays are not introduced during the inference execution of the considered CNN models.

Figure 6.7 shows the normalized energy consumption of the SRAM scratchpad versus the STT-MRAM one; the energy is broken down into leakage and dynamic energy. The

STT-MRAM scratchpad is way more energy efficient than the SRAM one, depending on the CNN model; up to 57.8× more energy efficient for Nasnet. The reason for this is the higher static power of SRAM, which accounts for most of the overall power consumption.

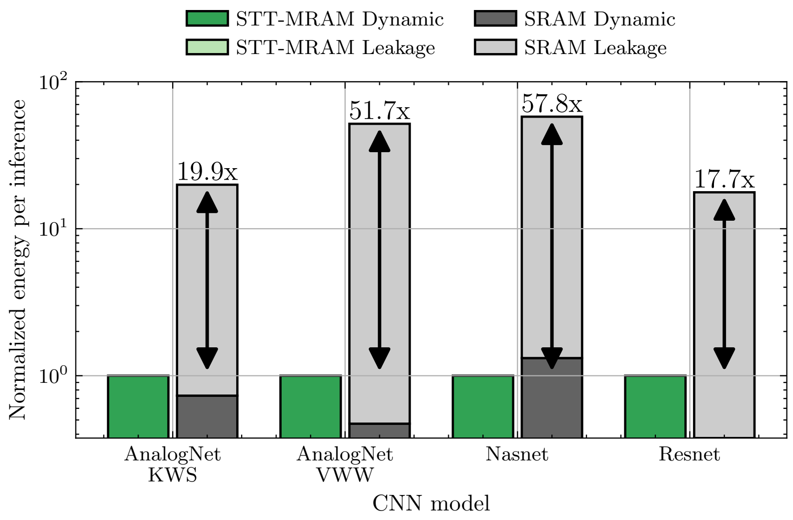


***Figure** 6.7: Normalized energy (in log scale) per inference for each CNN model. The scratchpad memory is implemented using SRAM technology versus the original one, implemented using STT-MRAM one. The energy is normalized to that of the STT-MRAM scratchapd and, in turn, broken down into dynamic and leakage energy.*

### 6.5.3 Depthwise-Convolution Analog vs. Digital

During this research and in line with findings from previous studies [138], limitations with depthwise convolutions were encountered. Such bottlenecks are critically notable when NasNet was deployed on the multi-tile architecture. Compared to Resnet50, which has 5× as many parameters (see Table 6.2), Nasnet requires more than twice as many tiles to be deployed, but reduces significantly the computation utilization of the arrays (8.3% in NasNet versus 97.6% in Resnet50). Out of the scope of this work, but highlighted by it, our results underscore a crucial need to adapt the NAS algorithm to the specific requirements and constraints of analog tile-based systems [138].

## 6.6 Concluding remarks

In this study, we have effectively addressed the primary bottlenecks in the MNEMOSENE architecture. By enhancing the original framework, we developed a robust multi-tile architecture capable of improved parallel processing and optimized buffering. Moreover, we successfully refined the communication between the tiles and the scratchpads, providing efficient data handling and significant performance enhancement. The architectural enhancements served to ensure efficient computation and storage integration, thereby propelling the MNEMOSENE design forward.

In addition, a thorough evaluation involving a wide range of neural networks and four different NVM technologies was conducted. These evaluations revealed a marked discrepancy in energy consumption across the different technologies. Amidst these, VGSOT-MRAM emerged as a promising compromise. It not only offers reduced area and lower energy requirements, but also presents a high degree of integration and speed, thus standing superior to STT-MRAM, and to both ReRAM and PCM. This finding accentuates VGSOT-MRAM's potential to effectively bridge the gap between computational efficiency and energy conservation in future technological advancements.

# Conclusions and future work

*Wars end.*

Detective Ellis Carter, The Wire.

# 7 Conclusions and future work

*This thesis has sought to further pave the way for the incorporation of NVM technologies down to the cache memory hierarchy. In this sense, this chapter sums up the conclusions of this dissertation and suggests new research lines to push NVM technologies towards commercial memory systems.*

## 7.1 Conclusions

The low endurance to repeated write operations is one of the main stumbling blocks preventing NVM technologies from being implemented in large-scale computer systems. The main contributions of this dissertation are aimed to fight against this limitation with the objective of designing long-lasting and fault-tolerant NVM-based caches.

The first part of this dissertation proposes microarchitectural solutions to optimize NVM-based LLCs for both performance and lifetime. First, we introduce L2C2, a new fault-tolerant NV-LLC design that combines fine-grain disabling (at byte granularity), data compression and an intra-frame wear-leveling. L2C2 leverages data compression not only to reduce the bytes written to the data array but to also allow partially defective frames to hold compressed blocks. The VLSI implementation of the block rearrangement circuitry shows that combining the intra-frame wear-leveling and the fine-grain disabling is feasible. This synergy enables to evenly distribute the write wear across the remaining non-faulty bytes within a frame. Besides, we propose L2C2+N, the redundant version of L2C2. Our mechanisms are transparent to the N-byte extension of the cache frames; distributing the write wear across the redundant bytes as well. The results show that L2C2+N is able to further extend the time during which the NV-LLC remains with near-peak nominal performance.

L2C2 designs can be augmented by incorporating SRAM in the data array, turning the original L2C2 architecture into a traditional hybrid LLC in which the NVM part of the data array is provided with the aforementioned fault-tolerant mechanisms. Existing hybrid LLC proposals particularly optimize for LLC lifetime by conservatively inserting cache blocks into the NVM part. These lifetime-focused optimizations significantly reduce the LLC performance. In this regard, Chapter 3 leverages L2C2 to address these performance and lifetime disparities seen in the state-of-the-art. Specifically, we propose new insertion policies for hybrid LLCs that optimize for both performance and lifetime. Our solution steers cache blocks towards either the SRAM or the NVM part considering the read/write-reuse and compression properties of the blocks. Besides, a threshold-based mechanism tunes the write-traffic to the NVM part by allowing more or less blocks to be allocated. This mechanism successfully captures the runtime behaviour of the workload and allows to further balance the tradeoff between performance and lifetime of the hybrid LLC.

The second part of this dissertation lays the methodological groundwork. First, Chapter 4 introduces a procedure that allows to forecast in detail the temporal evolution of NVM-based LLCs whose effective capacity gets lessened due to write operations. This procedure combines simulation phases, in which figures of interest (write rate to NVM frames, system IPC, etc.) are collected, with prediction phases that compute the next NVM bitcells that become faulty due to write operations. This procedure is comprehensively validated and effective to identify the limitations of state-of-the-art proposals in terms of lifetime and performance. Moreover, we believe that knowing the evolution of the memory capacity or the system performance could be pivotal for manufacturers to be confident in guaranteeing certain performance of their memory devices.

Besides, Chapter 5 introduces HyCSim, a trace-driven simulator that allowed us to do a fast design space exploration of hybrid LLCs insertion policies. HyCSim is open source, includes state-of-the-art insertion policies and is provided with the fault-tolerant mechanisms proposed in this thesis.

The third part of this dissertation shows the potential of NVM technologies in the CiM paradigm. In order to reduce the data movement, simple arithmetic operations can be performed directly within the memory array in a very efficient way. MNEMOSENE has been chosen as it is a fair example of a general-purpose CiM architecture based on a crossbar array of resistive NVM cells. By defining an abstract ISA, it can bridge the gap between high-level programming languages and the design of the underlying circuits. We showed that there is room for improvement in the original single-tile MNEMOSENE microarchitecture by identifying some buffering bottlenecks. Besides, a single-tile microarchitecture has inherent limitations in terms of scalability. Therefore, we scaled out the original single-tile by designing a multi-tile microarchitecture. This design comprises both a shared scratchpad memory and a communication framework to enable a seamless communication between tiles. Our comprehensive evaluation highlights the effectiveness of our proposals and the potential of VGSOT-MRAM technology, outperforming other NVMs in terms of energy efficiency for the selected CNN models.

## 7.2 Future work

This dissertation unveils future research to keep on pushing NVM technologies down in the memory hierarchy.

We developed a novel forecasting procedure that enables the evaluation and exploration of different figures of merit of both the cache and the system. This forecasting procedure assumes a generic approach in which the endurance of NVM bitcells follow a normal distribution of mean $\mu = 10^k$ writes, see Section 1.1.4, but these distributions may vary significantly between technologies. Besides, asymmetry of the different switching transitions, i.e. $0 \longrightarrow 0$, $0 \longrightarrow 1$, $1 \longrightarrow 0$, $1 \longrightarrow 1$, might have different implications depending on the NVM technology [15]. Therefore, we believe that the forecasting procedure can be extended by adding the particularities of each technology.

Furthermore, the forecasting procedure enables a holistic evaluation of microarchitectural solutions and their potential synergies when addressing the NVM endurance challenge from different perspectives. For instance, it allows to delve into the interplays among sophisticated ECC mechanisms, the refinement of memory write access patterns, the cache replacement mechanism, the disabling granularity of defective memory regions, etc. Although these approaches seem orthogonal, their combined analysis may lead to the discovery of novel designs and optimization strategies.

NVMs within the CiM paradigm has shown a great potential enhancing the energy efficiency of modern computing systems by bringing computing operations closer to the memory. However, the number of write operations in some of the CiM paradigms that use NVMs drastically increases compared to traditional von Neumann architectures [104]. Besides, the access pattern of CiM applications significantly changes. That is, not only are there more write operations, but the wear is more difficult to balance since the incorporation of traditional wear-leveling mechanisms is not straightforward. This novel paradigm opens up a variety of endurance problems that are yet to be solved; including wear-leveling and fault-tolerant mechanisms aimed at CiM devices and accelerators.

# 7 Conclusiones y trabajo futuro

*Esta tesis ha tratado de allanar aún más el camino para la incorporación de las tecnologías NVM hasta la jerarquía de memoria caché. En este sentido, este capítulo resume las conclusiones de esta tesis y sugiere nuevas líneas de investigación para seguir impulsando las tecnologías NVM hacia los sistemas de memoria comerciales.*

## 7.1 Conclusiones

La baja resistencia (*endurance*) a repetidas operaciones de escritura es uno de los principales escollos que impiden la implementación a gran escala de las tecnologías NVM en los sistemas computacionales. Las principales aportaciones de esta tesis pretenden luchar contra esta limitación coon el objetivo de diseñar cachés duraderas y tolerantes a fallos implementadas con tecnologías NVM.

La primera parte de esta tesis propone soluciones de microarquitectura para optimizar el rendimiento y el tiempo de vida útil de LLCs implementadas con NVMs. En primer lugar, presentamos L2C2, un nuevo diseño de NV-LLC tolerante a fallos que combina la deshabilitación fina de regiones de memoria (a nivel de byte), la compresión de datos y un wear-leveling para los contenedores de caché. L2C2 aprovecha la compresión no solo para reducir los bytes que se escriben en el array de datos, sino también para permitir que contenedores de caché parcialmente defectuosos alberguen bloques comprimidos. La implementación VLSI del circuito de reorganización de bloque muestra que la combinación del mecanismo de wear-leveling y de la deshabilitación con granularidad fina es factible. Esta sinergia permite distribuir de manera equitativa el desgaste que producen las operaciones de escritura entre los bytes operativos de un contenedor de caché. Además, proponemos L2C2+N, la versión redundante de L2C2. Nuestros mecanismos son transparentes a la extensión de N bytes de los contenedores de caché, distribuyendo el desgaste entre los bytes redundantes también. Los resultados muestran que L2C2+N es capaz de prolongar aún más el tiempo durante el cual la NV-LLC permancece con las prestaciones cercanas a las máximas nominales.

Los diseños de L2C2 pueden complementarse incorporando la implementación de una parte del array de datos con tecnología SRAM, convirtiendo la arquitectura L2C2 original en una LLC híbrida tradicional en la que la parte NVM del array de datos está dotada de los mecanismos tolerantes a fallos anteriormente mencionados. Las propuestas de LLCs híbridas del estado del arte optimizan el tiempo de vida útil de la LLC insertando de forma conservadora bloques de caché en la parte NVM. Estas optimizaciones orientadas a tiempo de vida útil reducen significativamente el rendimiento de la LLC híbrida. En este sentido, el Capítulo 3 aprovecha L2C2 para abordar estas disparidades entre el rendimiento y el tiempo de vida útil de las propuestas del estado del arte. En concreto, proponemos nuevas políticas de inserción para LLCs híbridas que optimizan tanto el rendimiento como el tiempo de vida útil. Nuestra propuesta direcciona los bloques de caché hacia la parte SRAM o la parte NVM teniendo en cuenta las propiedades de reutilización de lectura y escritura, y la compresibilidad de los bloques. Además, un mecanismo basado en umbrales ajusta el tráfico de escritura a la parte NVM permitiendo que se inserten más o menos bloques. Este mecanismo captura con éxito el comportamiento en tiempo de ejecución de la carga de trabajo y permite equilibrar aún más el compromiso entre rendimiento y tiempo de vida útil de la LLC híbrida.

La segunda parte de la tesis sienta las bases metodoógicas. En primer lugar, el Capítulo 4 introduce un procedimiento que permite pronosticar minuciosamente la evolución temporal de las LLC implementadas con NVMs cuya capacidad efectiva se ve mermada debido a als operaciones de escritura. Este procedimiento combina fases de simulación, en las que se recogen cifras de interés (velocidad de escritura en los contenedores de caché, IPC del sistema, etc.), con fases de predicción, que calculan las siguientes celdas que se convertirán en defectuosas debido a las escrituras. Este procedimiento se valida exhaustivamente y es eficaz para identificar las limitaciones de las propuestas del estado del arte en términos de tiempo de vida útil y rendimiento. Además, creemos que conocer la evolución de la capacidad de la memoria caché o el rendimiento del sistema podría ser fundamental para que los fabricantes se sientan seguros a la hora de garantizar ciertas prestaciones de sus dispositivos de memoria.

Además, el Capítulo 5 presenta HyCSim, un simulador basado en trazas de memoria que ha permitido agilizar la exploración del espacio de diseño de las políticas de inserción para LLCs híbridas. HyCSim es de código abierto, incluye las políticas de inserción del estado del arte, y está provisto de los mecanismos de tolerancia a fallos porpuestos en esta tesis.

La tercera parte de esta tesis expone el potencial de las tecnologías NVM en el paradigma CiM. Para reducir el movimiento de datos, algunas operaciones aritméticas sencillas pueden realizarse directamente dentro del array de memoria muy eficientemente. Se ha escogido

MNEMOSENE ya que es un buen ejemplo de arquitectura CiM de propósito general basado en un crossbar array de celdas NVM resistivas. Definiendo una ISA abstracta, puede servir de nexo de unión entre los lenguajes de programación de alto nivel y el diseño de los circuitos electrónicos subyacentes. En esta tesis se demuestra que hay margen de mejora en la microarquitectura MNEMOSENE de tile único mediante la identificación de algunos cuellos de botella en los búferes de almacenamiento. Además, una microarquitectura de un solo tile tiene limitaciones inherentes en términos de escalabilidad. Por lo tanto, hemos ampliado la microarquitectura original de un solo tile diseñando una microarquitectura de varios tiles. Este diseño incluye tanto una memoria scratchpad compartida como un sistema de interconexión que permiten una comunicación fluida entre los tiles. La evaluación detallada pone de manifiesto la utilidad de las propuestas y el potencial de la tecnología VGSOT-MRAM, superando a otras NVMs en términos de eficiencia energética para los modelos CNN seleccionados.

## 7.2 Trabajo futuro

Esta tesis desvela investigaciones de cara al futuro para seguir impulsando la adopción de las tecnologías NVM en la jerarquía de memoria.

Se ha desarrollado un novedoso procedimiento de pronóstico que permite evaluar y explorar diferentes índices de interés de la caché como del sistema. Este procedimiento de pronóstico parte de un enfoque genérico en el que la resistencia de las celdas NVM sigue una distribución normal de media $\mu = 10^k$ escrituras, véase la Sección 1.1.4, pero estas distribuciones pueden variar significativamente entre tecnologías. Además, la asimetría entre las diferentes transiciones de conmutación, es decir, $0 \rightarrow 0$, $0 \rightarrow 1$, $1 \rightarrow 0$, $1 \rightarrow 1$, podría tener diferentes implicaciones en función de la tecnología NVM [15]. Por lo tanto, creemos que el procedimiento de previsión puede ampliarse añadiendo las particularidades de cada tecnología.

Además, el procedimiento de pronóstico permite una evaluación exhaustiva de las soluciones microarquitectónicas y sus posibles sinergias a la hora de abordar el reto del problema de la endurance de las NVMs desde diferentes perspectivas. Por ejemplo, permite profundizar en las interacciones entre mecanismos de ECC sofisticados, el refinamiento de los patrones de escritura en la memoria, los mecanismos de reemplazo de cache, la granularidad de la deshabilitación de regiones defectuosas de memoria, etc. Aunque estos enfoques parezcan ortogonales, su análisis combinado puede descubrir nuevos diseños y estrategias de optimización.

Las NVMs dentro del paradigma CiM han demostrado un gran potencial mejorando la eficiencia energética de los sistemas de computación modernos al acercar las operaciones de computación a la memoria. Sin embargo, el número de operaciones de escritura en algunos paradigmas CiM que usan NVMs aumenta drásticamente en comparación con las arquitecturas von Neumann tradicionales [104]. Además, el patrón de acceso de las aplicaciones CiM varía significativamente. Es decir, no solo hay más operaciones de escritura, sino que el desgaste es más difícil de equilibrar, ya que la incorporación de los mecanismos tradicionales de nivelación del desgaste no es sencilla. Este novedoso paradigma abre una variedad de problemas de endurance que todavía están por resolver;

entre ellos, los mecanismos de nivelación de desgaste y tolerancia a fallos orientados a dispositivos y aceleradores CiM.

# Appendices

# A

# Time scaling of forecasted indexes when considering bitcells with more endurance.

Let

$$N(w_b;\ \mu,\ \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(w_b-\mu)^2}{2\sigma^2}} \tag{A.1}$$

be the normal probability distribution function that estimates the number of writes $w_b$ causing failure in a *baseline* bitcell. Assuming a constant write rate WR (writes/s) on the baseline bitcell, the probability distribution function of the failure time $t_b$ can be obtained from Eq. A.1 by the linear transformation $t_b = \frac{w_b}{\mathrm{WR}}$:

$$N(t_b;\ \frac{\mu}{\mathrm{WR}},\ \frac{\sigma}{\mathrm{WR}}) \tag{A.2}$$

We can characterize an *improved* bitcell, with $k$ times higher endurance, by applying to Eq. A.1 and A.2 the linear transformation $w_i = w_b \cdot k$. Thus, the probability distribution functions of the number of writes and failure time for the improved bitcell are, respectively:

$$N(w_i;\ \mu \cdot k,\ \ \sigma \cdot k) \quad \text{and} \quad N(t_i;\ \frac{\mu \cdot k}{\mathrm{WR}},\ \frac{\sigma \cdot k}{\mathrm{WR}}) \tag{A.3}$$

On the other hand, the probability of failure of the baseline bitcell, $P_b$, at a time $t_b \leq t$ is:

$$P_b(t_b \leq t) = \int_0^t N(t_b;\ \frac{\mu}{\mathrm{WR}},\ \frac{\sigma}{\mathrm{WR}}) dt_b \tag{A.4}$$

To know the probability of failure of the improved bitcell, $P_i$, at a time $t_i \leq t$ from $P_b$, it is necessary to apply another linear transformation: $t_b = \frac{t_i}{k}$. Thus:

$$P_i(t_i \leq t) = P_b(t_b \leq \text{lin\_trans}_{i \to b}(t)) = P_b(t_b \leq \frac{t}{k})) \tag{A.5}$$

Rewriting the two probabilities as a function of $t$, we have:

$$P_i(t) = P_b(\frac{t}{k}) \tag{A.6}$$

To conclude, let us consider a cache with $c$ baseline bitcells, each with an endurance approximated by the probability distribution of Eq. A.1 and subjected to a constant per-cell write rate WR (writes/s). Assuming bit granularity the decrease of its effective capacity with time, $\text{Ceff}_b(t)$ is:

$$\text{Ceff}_b(t) = C \cdot (1 - P_b(t)) \tag{A.7}$$

And for a cache of the same size made with improved bitcells:

$$\text{Ceff}_i(t) = C \cdot (1 - P_b(\frac{t}{k})) \tag{A.8}$$

In this case, with byte granularity and different write rates in each frame, it can be reasoned in the same way. That is, any forecasted index with enhanced cells at time t matches the same index forecasted with base cells but at time $\frac{t}{k}$.

# Bibliography

[1] Bulent Abali, Bart Blaner, John Reilly, Matthias Klein, Ashutosh Mishra, Craig B. Agricola, Bedri Sendir, Alper Buyuktosunoglu, Christian Jacobi, William J. Starke, Haren Myneni, and Charlie Wang. Data compression accelerator on ibm power9 and z15 processors : Industrial product. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 1–14, 2020.

[2] Sukarn Agarwal. Linovo: Longevity enhancement of non-volatile caches by placement, write-restriction & victim caching in chip multi-processors. In *PhD Dissertation*, Guwahati, India, 2020.

[3] Junwhan Ahn, Sungjoo Yoo, and Kiyoung Choi. Selectively protecting error-correcting code for area-efficient and reliable stt-ram caches. In *2013 18th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 285–290. IEEE, 2013.

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

[5] Junwhan Ahn, Sungjoo Yoo, and Kiyoung Choi. Prediction hybrid cache: An energy-efficient stt-ram cache architecture. *IEEE Transactions on Computers*, 65(3):940–951, 2015.

[6] Tanner Andrulis, Joel S Emer, and Vivienne Sze. Raella: Reforming the arithmetic for efficient, low-resolution, and low-loss analog pim: No retraining required! In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, pages 1–16, 2023.

[7] Dmytro Apalkov, Alexey Khvalkovskiy, Steven Watts, Vladimir Nikitin, Xueti Tang, Daniel Lottis, Kiseok Moon, Xiao Luo, Eugene Chen, Adrian Ong, et al. Spin-transfer torque magnetic random access memory (stt-mram). *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 9(2):1–35, 2013.

[8] Lotfi Belkhir and Ahmed Elmeligi. Assessing ict global emissions footprint: Trends to 2040 & recommendations. *Journal of cleaner production*, 177:448–463, 2018.

[9] Koustav Bhattacharya, Nagarajan Ranganathan, and Soontae Kim. A framework for correction of multi-bit soft errors in l2 caches based on redundancy. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 17(2):194–206, 2008.

[10] Amine Bouaouda, Karim Afdel, and Rachida Abounacer. Forecasting the energy consumption of cloud data centers based on container placement with ant colony optimization and bin packing. In *2022 5th Conference on Cloud and Internet of Things (CIoT)*, pages 150–157, 2022.

[11] Hadi Brais, Rajshekar Kalayappan, and Preeti Ranjan Panda. A survey of cache simulators. *ACM Comput. Surv.*, 53(1), feb 2020.

[12] James Bucek, Klaus-Dieter Lange, and Jóakim v. Kistowski. Spec cpu2017: Next-generation compute benchmark. In *Companion of the 2018 ACM/SPEC Int. Conf. on Performance Engineering*, ICPE '18, page 41–42, New York, NY, USA, 2018. Association for Computing Machinery.

[13] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience*, 41(1):23–50, 2011.

[14] Roberto Carboni. Characterization and modeling of spin-transfer torque (stt) magnetic memory for computing applications. In *Special Topics in Information Technology*, pages 51–62. Springer, Cham, 2021.

[15] Roberto Carboni, Stefano Ambrogio, Wei Chen, Manzar Siddik, Jon Harms, Andy Lyle, Witold Kula, Gurtej Sandhu, and Daniele Ielmini. Modeling of breakdown-limited endurance in spin-transfer torque magnetic memory under pulsed cycling regime. *IEEE Transactions on Electron Devices*, 65(6):2470–2478, 2018.

[16] Trevor E. Carlson, Wim Heirman, and Lieven Eeckhout. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 52:1–52:12, New York, NY, USA, 2011. ACM.

[17] Jonathan Chang, Ming Huang, Jonathan Shoemaker, John Benoit, Szu-Liang Chen, Wei Chen, Siufu Chiu, Raghuraman Ganesan, Gloria Leong, Venkata Lukka, Stefan Rusu, and Durgesh Srivastava. The 65-nm 16-mb shared on-die l3 cache for the dual-core intel xeon processor 7100 series. *IEEE Journal of Solid-State Circuits*, 42(4):846–852, 2007.

[18] Mu-Tien Chang, Shih-Lien Lu, and Bruce Jacob. Impact of cache coherence protocols on the power consumption of stt-ram-based llc. In *The Memory Forum Workshop*, 2014.

[19] Hsiang-Yun Cheng, Jishen Zhao, Jack Sampson, Mary Jane Irwin, Aamer Jaleel, Yu Lu, and Yuan Xie. Lap: Loop-block aware inclusion properties for energy-efficient asymmetric last level caches. In *2016 ACM/IEEE 43rd Ann. Int. Symp. on Computer Architecture (ISCA)*, pages 103–114, 2016.

[20] Elham Cheshmikhani, Hamed Farbeh, and Hossein Asadi. Enhancing reliability of stt-mram caches by eliminating read disturbance accumulation. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 854–859, 2019.

[21] Elham Cheshmikhani, Hamed Farbeh, and Hossein Asadi. A system-level framework for analytical and empirical reliability exploration of stt-mram caches. *IEEE Transactions on Reliability*, 69(2):594–610, 2020.

[22] 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.

[23] Elham Cheshmikhani, Hamed Farbeh, Seyed Ghassem Miremadi, and Hossein Asadi. Ta-lrw: A replacement policy for error rate reduction in stt-mram caches. *IEEE Transactions on Computers*, 68(3):455–470, 2019.

[24] Yu-Der Chih, Yi-Chun Shih, Chia-Fu Lee, Yen-An Chang, Po-Hao Lee, Hon-Jarn Lin, Yu-Lin Chen, Chieh-Pu Lo, Meng-Chun Shih, Kuei-Hung Shen, Harry Chuang, and Tsung-Yung Jonathan Chang. 13.3 a 22nm 32mb embedded stt-mram with 10ns read speed, 1m cycle write endurance, 10 years retention at 150° c and high immunity to magnetic field interference. In *2020 IEEE Int. Solid-State Circuits Conf. (ISSCC)*, pages 222–224. IEEE, 2020.

[25] Sangyeun Cho and Hyunjin Lee. Flip-n-write: A simple deterministic technique to improve pram write performance, energy and endurance. In *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 347–357, 2009.

[26] Ju Hee Choi, Jong Wook Kwak, Seong Tae Jhang, and Chu Shik Jhon. Adaptive cache compression for non-volatile memories in embedded system. In *Proc. of the 2014 Conf. on Research in Adaptive and Convergent Systems*, pages 52–57, 2014.

[27] Ju-Hee Choi and Gi-Ho Park. Nvm way allocation scheme to reduce nvm writes for hybrid cache architecture in chip-multiprocessors. *IEEE Trans. on Parallel and Distributed Systems*, 28(10):2896–2910, 2017.

[28] Marcelo Cintra and Niklas Linkewitsch. Characterizing the impact of process variation on write endurance enhancing techniques for non-volatile memory systems. In *Proc. of the ACM SIGMETRICS/Int. Conf. on Measurement and modeling of computer systems*, pages 217–228, 2013.

[29] Jeanine Cook, Jonathan Cook, and Waleed Alkohlani. A statistical performance model of the opteron processor. *SIGMETRICS Perform. Eval. Rev.*, 38(4):75–80, mar 2011.

[30] R.H. Dennard, F.H. Gaensslen, Hwa-Nien Yu, V.L. Rideout, E. Bassous, and A.R. LeBlanc. Design of ion-implanted mosfet's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, 1974.

[31] D. B. Dgien, P. M. Palangappa, N. A. Hunter, J. Li, and K. Mohanram. Compression architecture for bit-write reduction in non-volatile memory technologies. In *2014 IEEE/ACM Int. Symp. on Nanoscale Architectures (NANOARCH)*, pages 51–56. IEEE, 2014.

[32] Haiwei Dong, Ali Munir, Hanine Tout, and Yashar Ganjali. Next-generation data center network enabled by machine learning: Review, challenges, and opportunities. *IEEE Access*, 9:136459–136475, 2021.

[33] Xiangyu Dong, Cong Xu, Yuan Xie, and Norman P Jouppi. Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 31(7):994–1007, 2012.

[34] Jan Edler and Mark D. Hill. Dinero iv: Trace-driven uniprocessor cache simulator. `http://pages.cs.wisc.edu/~markhill/DineroIV/`, 1994.

[35] Carlos Escuin. Hycsim code. `https://gitlab.com/uz-gaz/hycsim`, 2022.

[36] Carlos Escuin. Forecasting code. `https://gitlab.com/uz-gaz/l2c2-forecasting`, 2023.

[37] Carlos Escuin, Fernando García-Redondo, Mahdi Zahedi, Pablo Ibánez, Teresa Monreal, Víctor Viñals, José M. Llabería, James Myers, Julien Ryckaert, Dwaipayan Biswas, and Francky Catthoor. Leveraging data compression for performance-efficient and long-lasting nvm-based last-level caches. In *Submitted to 2023 30th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE, 2023.

[38] Carlos Escuin, Pablo Ibañez, Denis Navarro, Teresa Monreal, Jose M Llaberia, and Victor Viñals. L2c2: Last-level compressed-cache nvm and a procedure to forecast performance and lifetime. *Plos one*, 18(2):e0278346, 2023.

[39] Carlos Escuin, Asif Ali Khan, Pablo Ibánez, Teresa Monreal, Jeronimo Castrillon, and Víctor Viñals. Compression-aware and performance-efficient insertion policies for long-lasting hybrid llcs. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 179–192. IEEE, 2023.

[40] Carlos Escuin, Asif Ali Khan, Pablo Ibánez, Teresa Monreal, Denis Navarro, José M. Llabería, Jeronimo Castrillon, and Víctor Viñals. Leveraging data compression for performance-efficient and long-lasting nvm-based last-level caches. In *14th Annual Non-Volatile Memory Workshop*. University of Califronia San Diego, 2023.

[41] Carlos Escuin, Asif Ali Khan, Pablo Ibañez, Teresa Monreal, Victor Viñals, and Jeronimo Castrillon. Hycsim: A rapid design space exploration tool for emerging hybrid last-level caches. In *System Engineering for constrained embedded systems (DroneSE and RAPIDO '22)*, pages 1–6, New York, NY, USA, 2022. ACM.

[42] Carlos Escuín Blasco, Teresa Monreal Arnal, José M Llaberia Griñó, Victor Viñals Yúfera, and Pablo Ibáñez Marín. Stt-ram memory hierarchy designs aimed to performance, reliability and energy consumption. In *ACACES 2019: July 17, 2019, Fiuggi, Italy: poster abstracts*, pages 231–234. European Network of Excellence on High Performance and Embedded Architecture ..., 2019.

[43] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, page 365–376, New York, NY, USA, 2011. Association for Computing Machinery.

[44] Hamed Farbeh, Hyeonggyu Kim, Seyed Ghassem Miremadi, and Soontae Kim. Floating-ecc: Dynamic repositioning of error correcting code bits for extending the lifetime of stt-ram caches. *IEEE Trans. on Computers*, 65(12):3661–3675, 2016.

[45] Alexandra Ferrerón, Dario Suárez-Grácia, Jesús Alastruey-Benedé, Teresa Monreal-Arnal, and Pablo Ibáñez. Concertina: Squeezing in cache content to operate at near-threshold voltage. *IEEE Trans. on Computers*, 65(3):755–769, 2015.

[46] Fernando García Redondo. *Resistive RAM: simulation and modeling for reliable design.* PhD thesis, Telecomunicacion, 2017.

[47] Fernando García-Redondo et al. Stt-mram stochastic and defects-aware dtco for last level cache at advanced process nodes. In *IEEE 53st European Solid-State Device Research Conference (ESSDERC)*, 2023.

[48] O. Golonzka, J. G. Alzate, U. Arslan, M. Bohr, P. Bai, J. Brockman, B. Buford, C. Connor, N. Das, B. Doyle, T. Ghani, F. Hamzaoglu, P. Heil, P. Hentges, R. Jahan, D. Kencke, B. Lin, M. Lu, M. Mainuddin, M. Meterelliyoz, P. Nguyen, D. Nikonov, K. O'brien, J.O Donnell, K. Oguz, D. Ouellette, J. Park, J. Pellegren, C. Puls, P. Quintero, T. Rahman, A. Romang, M. Sekhar, A. Selarka, M. Seth, A. J. Smith, A. K. Smith, L. Wei, C. Wiegand, Z. Zhang, and K. Fischer. Mram as embedded non-volatile memory solution for 22ffl finfet technology. In *2018 IEEE Int. Electron Devices Meeting (IEDM)*, pages 18–1. IEEE, 2018.

[49] Fazal Hameed and Jeronimo Castrillon. A novel hybrid dram/stt-ram last-level-cache architecture for performance, energy, and endurance enhancement. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(10):2375–2386, 2019.

[50] Julian Hammer. Pycachesim: A single-core cache hierarchy simulator in python. https://github.com/RRZE-HPC/pycachesim, 2016.

[51] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: Efficient inference engine on compressed deep neural network. *ACM SIGARCH Computer Architecture News*, 44(3):243–254, 2016.

[52] Alexander Hankin, Tomer Shapira, Karthik Sangaiah, Michael Lui, and Mark Hempstead. Evaluation of non-volatile memory based last level cache given modern use case behavior. In *2019 IEEE International Symposium on Workload Characterization (IISWC)*, pages 143–154. IEEE, 2019.

[53] Alexander Hardtdegen, Camilla La Torre, Felix Cüppers, Stephan Menzel, Rainer Waser, and Susanne Hoffmann-Eifert. Improved switching stability and the effect of an internal series resistor in hfo 2/tio x bilayer reram cells. *IEEE transactions on electron devices*, 65(8):3229–3236, 2018.

[54] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[55] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011.

[56] John L Henning. Spec cpu2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News*, 34(4):1–17, 2006.

[57] Seokin Hong, Bulent Abali, Alper Buyuktosunoglu, Michael B Healy, and Prashant J Nair. Touché: Towards ideal and efficient cache compression by mitigating tag area overheads. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 453–465, 2019.

[58] Yiming Huai. Spin-transfer torque mram (stt-mram): Challenges and prospects. *AAPPS bulletin*, 18(6):33–40, 2008.

[59] P Huang, B Chen, YJ Wang, FF Zhang, L Shen, R Liu, L Zeng, G Du, X Zhang, B Gao, et al. Analytic model of endurance degradation and its practical applications for operation scheme optimization in metal oxide based rram. In *2013 IEEE International electron devices meeting*, pages 22–5. IEEE, 2013.

[60] Engin Ipek, Jeremy Condit, Edmund B Nightingale, Doug Burger, and Thomas Moscibroda. Dynamically replicated memory: building reliable systems from nanoscale resistive memories. *ACM Sigplan Notices*, 45(3):3–14, 2010.

[61] Ravi Iyer. On modeling and analyzing cache hierarchies using casper. In *11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems, 2003. MASCOTS 2003.*, pages 182–187. IEEE, 2003.

[62] Joseph Izraelevitz, Jian Yang, Lu Zhang, Juno Kim, Xiao Liu, Amirsaman Memaripour, Yun Joon Soh, Zixuan Wang, Yi Xu, Subramanya R Dulloor, et al. Basic performance measurements of the intel optane dc persistent memory module. *arXiv preprint arXiv:1903.05714*, 2019.

[63] A. Jadidi, M. Arjomand, M. K. Tavana, D. R. Kaeli, M. T. Kandemir, and C. R. Das. Exploring the potential for collaborative data compression and hard-error tolerance in pcm memories. In *2017 47th Ann. IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN)*, pages 85–96, 2017.

[64] Aamer Jaleel, Robert S Cohn, Chi-Keung Luk, and Bruce Jacob. Cmp$im: A pin-based on-the-fly multi-core cache simulator. In *Proceedings of the Fourth Annual Workshop on Modeling, Benchmarking and Simulation (MoBS), co-located with ISCA*, pages 28–36, 2008.

[65] 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 Conf. & Exhibition (DATE 2010)*, pages 136–141. IEEE, 2010.

[66] Wang Kang, WeiSheng Zhao, Zhaohao Wang, Yue Zhang, Jacques-Olivier Klein, Youguang Zhang, Claude Chappert, and Dafiné Ravelosona. A low-cost built-in error correction circuit design for stt-mram reliability improvement. *Microelectronics Reliability*, 53(9-11):1224–1229, 2013.

[67] Sachhidh Kannan, Jeyavijayan Rajendran, Ramesh Karri, and Ozgur Sinanoglu. Sneak-path testing of crossbar-based nonvolatile random access memories. *IEEE Transactions on Nanotechnology*, 12(3):413–426, 2013.

[68] Riduan Khaddam-Aljameh, Milos Stanisavljevic, J Fornt Mas, Geethan Karunaratne, Matthias Braendli, Femg Liu, Abhairaj Singh, Silvia M Müller, Urs Egger, Anastasios Petropoulos, et al. Hermes core–a 14nm cmos and pcm-based in-memory compute core using an array of 300ps/lsb linearized cco-based adcs and local digital processing. In *2021 Symposium on VLSI Circuits*, pages 1–2. IEEE, 2021.

[69] Asif Ali Khan, Fazal Hameed, and Jeronimo Castrillon. Nvmain extension for multi-level cache systems. In *Proceedings of the Rapido'18 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*, RAPIDO '18, New York, NY, USA, 2018. Association for Computing Machinery.

[70] Tahseen Khan, Wenhong Tian, Shashikant Ilager, and Rajkumar Buyya. Workload forecasting and energy state estimation in cloud data centres: Ml-centric approach. *Future Generation Computer Systems*, 128:320–332, 2022.

[71] Beomjun Kim, Prashant J Nair, and Seokin Hong. Adam: Adaptive block placement with metadata embedding for hybrid caches. In *2020 IEEE 38th International Conference on Computer Design (ICCD)*, pages 421–424. IEEE, 2020.

[72] Jangwoo Kim, Nikos Hardavellas, Ken Mai, Babak Falsafi, and James Hoe. Multi-bit error tolerant caches using two-dimensional error coding. In *40th Ann. IEEE/ACM Int. Symp. on Microarchitecture (MICRO 2007)*, pages 197–209, 2007.

[73] Namhyung Kim, Junwhan Ahn, Woong Seo, and Kiyoung Choi. Energy-efficient exclusive last-level hybrid caches consisting of sram and stt-ram. In *2015 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 183–188. IEEE, 2015.

[74] 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 Ann. Int. Symp. on Computer Architecture (ISCA)*, pages 315–327. IEEE, 2018.

[75] Aleksey S. Kozhin and Alexander V. Surchenko. Evaluation of cache compression for elbrus processors. In *2018 Engineering and Telecommunication (EnT-MIPT)*, pages 135–139, 2018.

[76] Manuel Le Gallo, Abu Sebastian, Giovanni Cherubini, Heiner Giefers, and Evangelos Eleftheriou. Compressed sensing with approximate message passing using in-memory computing. *IEEE Transactions on Electron Devices*, 65(10):4304–4312, 2018.

[77] Benjamin C Lee, Engin Ipek, Onur Mutlu, and Doug Burger. Architecting phase change memory as a scalable dram alternative. In *Proc. of the 36th annual Int. Symp. on Computer architecture*, pages 2–13, 2009.

[78] Benjamin C. Lee, Ping Zhou, Jun Yang, Youtao Zhang, Bo Zhao, Engin Ipek, Onur Mutlu, and Doug Burger. Phase-change technology and the future of main memory. *IEEE Micro*, 30(1):143–143, 2010.

[79] Yong Kyu Lee, Yoonjong Song, JooChan Kim, SeChung Oh, Byoung-Jae Bae, SangHumn Lee, JungHyuk Lee, UngHwan Pi, Boyoung Seo, Hyunsung Jung, Kilho Lee, HyunChul Shin, Hyuntaek Jung, Mark Pyo, Artur Antonyan, Daesop Lee, Sohee Hwang, Daehyun Jang, Yongsung Ji, Seungbae Lee, Jungman Lim, Kwan-Hyeob Koh, Kihyun Hwang, Hyeongsun Hong, Kichul Park, Gitae Jeong, Jong Shik Yoon, and E.S. Jung. Embedded stt-mram in 28-nm fdsoi logic process for industrial mcu/iot application. In *2018 IEEE Symp. on VLSI Technology*, pages 181–182. IEEE, 2018.

[80] Habte Lejebo Leka, Zhang Fengli, Ayantu Tesfaye Kenea, Abebe Tamrat Tegene, Peter Atandoh, and Negalign Wake Hundera. A hybrid cnn-lstm model for virtual machine workload forecasting in cloud data center. In *2021 18th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*, pages 474–478, 2021.

[81] Keqin Li. Profit maximization in a federated cloud by optimal workload management and server speed setting. *IEEE Transactions on Sustainable Computing*, pages 1–1, 2021.

[82] Shuangchen Li, Alvin Oliver Glova, Xing Hu, Peng Gu, Dimin Niu, Krishna T Malladi, Hongzhong Zheng, Bob Brennan, and Yuan Xie. Scope: A stochastic computing engine for dram-based in-situ accelerator. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 696–709. IEEE, 2018.

[83] Jason Lowe-Power, Abdul Mutaal Ahmad, Ayaz Akram, Mohammad Alian, Rico Amslinger, Matteo Andreozzi, Adrià Armejach, Nils Asmussen, Brad Beckmann, Srikant Bharadwaj, Gabe Black, Gedare Bloom, Bobby R. Bruce, Daniel Rodrigues Carvalho, Jeronimo Castrillon, Lizhong Chen, Nicolas Derumigny, Stephan Diestelhorst, Wendy Elsasser, Carlos Escuin, Marjan Fariborz, Amin Farmahini-Farahani, Pouya Fotouhi, Ryan Gambord, Jayneel Gandhi, Dibakar Gope, Thomas Grass, Anthony Gutierrez, Bagus Hanindhito, Andreas Hansson, Swapnil Haria, Austin Harris, Timothy Hayes, Adrian Herrera, Matthew Horsnell, Syed Ali Raza Jafri, Radhika Jagtap, Hanhwi Jang, Reiley Jeyapaul, Timothy M. Jones, Matthias Jung, Subash Kannoth, Hamidreza Khaleghzadeh, Yuetsu Kodama, Tushar Krishna, Tommaso Marinelli, Christian Menard, Andrea Mondelli, Miquel Moreto, Tiago Mück, Omar Naji, Krishnendra Nathella, Hoa Nguyen, Nikos Nikoleris, Lena E. Olson, Marc Orr, Binh Pham, Pablo Prieto, Trivikram Reddy, Alec Roelke, Mahyar Samani, Andreas Sandberg, Javier Setoain, Boris Shingarov, Matthew D. Sinclair, Tuan Ta, Rahul Thakur, Giacomo Travaglini, Michael Upton, Nilay Vaish, Ilias Vougioukas, William Wang, Zhengrong Wang, Norbert Wehn, Christian Weis, David A. Wood, Hongil Yoon, and Éder F. Zulian. The gem5 simulator: Version 20.0+. *arXiv preprint arXiv:2007.03152*, 2020.

[84] Jing-Yuan Luo, Hsiang-Yun Cheng, Ing-Chao Lin, and Da-Wei Chang. Tap: Reducing the energy of asymmetric hybrid last-level cache via thrashing aware placement and migration. *IEEE Transactions on Computers*, 68(12):1704–1719, 2019.

[85] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. Statistical and machine learning forecasting methods: Concerns and ways forward. *PloS one*, 13(3):e0194889, 2018.

[86] Mayler Martins, Jody Maick Matos, Renato P Ribas, André Reis, Guilherme Schlinker, Lucio Rech, and Jens Michelsen. Open cell library in 15nm freepdk technology. In *Proceedings of the 2015 Symposium on International Symposium on Physical Design*, pages 171–178, 2015.

[87] Sparsh Mittal. Mitigating read disturbance errors in stt-ram caches by using data compression. In *Nanoelectronics: Devices, Circuits and Systems*, pages 133–152. Elsevier, 2019.

[88] Sparsh Mittal and Jeffrey S. Vetter. Reliability tradeoffs in design of volatile and nonvolatile caches. *Journal of Circuits, Systems and Computers*, 25(11):1650139, 2016.

[89] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8):114––117, 1965.

[90] Naveen Muralimanohar, Rajeev Balasubramonian, and Norm Jouppi. Optimizing nuca organizations and wiring alternatives for large caches with cacti 6.0. In *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*, pages 3–14. IEEE, 2007.

[91] Masanori Natsui, Akira Tamakoshi, Hiroaki Honjo, Toshinari Watanabe, Takashi Nasuno, Chaoliang Zhang, Takaho Tanigawa, Hirofumi Inoue, Masaaki Niwa, Toru Yoshiduka, Yasuo Noguchi, Mitsuo Yasuhira, Yitao Ma, Hui Shen, Shunsuke Fukami, Hideo Sato, Shoji Ikeda, Hideo Ohno, Tetsuo Endoh, and Takahiro Hanyu. Dual-port sot-mram achieving 90-mhz read

and 60-mhz write operations under field-assistance-free condition. *IEEE Journal of Solid-State Circuits*, 2020.

[92] Agustín Navarro-Torres, Jesús Alastruey-Benedé, Pablo Ibáñez-Marín, and Víctor Viñals-Yúfera. Memory hierarchy characterization of spec cpu2006 and spec cpu2017 on the intel xeon skylake-sp. *Plos one*, 14(8):e0220135, 2019.

[93] Nicholas Nethercote and Julian Seward. Valgrind: a framework for heavyweight dynamic binary instrumentation. *ACM Sigplan notices*, 42(6):89–100, 2007.

[94] Hiroki Noguchi, Kazutaka Ikegami, Satoshi Takaya, Eishi Arima, Keiichi Kushida, Atsushi Kawasumi, Hiroyuki Hara, Keiko Abe, Naoharu Shimomura, Junichi Ito, Shinobu Fujita, Takashi Nakada, and Hiroshi Nakamura. 7.2 4mb stt-mram-based cache with memory-access-aware power optimization and write-verify-write / read-modify-write scheme. In *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 132–133, 2016.

[95] Poovaiah M. Palangappa and Kartik Mohanram. Compex: Compression-expansion coding for energy, latency, and lifetime improvements in mlc/tlc nvm. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 90–101, 2016.

[96] Poovaiah M Palangappa and Kartik Mohanram. Castle: compression architecture for secure low latency, low energy, high endurance nvms. In *2018 55th ACM/ESDA/IEEE Design Automation Conf. (DAC)*, pages 1–6. IEEE, 2018.

[97] Eva Patel and Dharmender Singh Kushwaha. A hybrid cnn-lstm model for predicting server load in cloud computing. *The Journal of Supercomputing*, 78(8):1–30, 2022.

[98] David Patterson, Thomas Anderson, Neal Cardwell, Richard Fromm, Kimberly Keeton, Christoforos Kozyrakis, Randi Thomas, and Katherine Yelick. A case for intelligent ram. *IEEE micro*, 17(2):34–44, 1997.

[99] Gennady Pekhimenko, Vivek Seshadri, Onur Mutlu, Michael A Kozuch, Phillip B Gibbons, and Todd C Mowry. Base-delta-immediate compression: Practical data compression for on-chip caches. In *2012 21st Int. Conf. on Parallel Architectures and Compilation Techniques (PACT)*, pages 377–388. IEEE, 2012.

[100] M. Poremba, T. Zhang, and Y. Xie. Nvmain 2.0: A user-friendly memory simulator to model (non-)volatile memory systems. *IEEE Computer Architecture Letters*, 14(2):140–143, July 2015.

[101] Moinuddin K Qureshi, Sudhanva Gurumurthi, and Bipin Rajendran. Phase change memory: From devices to systems. *Synthesis Lectures on Computer Architecture*, 6(4):1–134, 2011.

[102] Moinuddin K Qureshi, Aamer Jaleel, Yale N Patt, Simon C Steely, and Joel Emer. Set-dueling-controlled adaptive insertion for high-performance caching. *IEEE micro*, 28(1):91–98, 2008.

[103] Moinuddin K Qureshi, Vijayalakshmi Srinivasan, and Jude A Rivers. Scalable high performance main memory system using phase-change memory technology. In *Proceedings of the 36th annual international symposium on Computer architecture*, pages 24–33, 2009.

[104] Salonik Resch, Husrev Cilasun, Zamshed Chowdhury, Masoud Zabihi, Zhengyang Zhao, Jian-Ping Wang, Sachin Sapatnekar, and Ulya R. Karpuzcu. On endurance of processing in (nonvolatile) memory. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ISCA ’23, New York, NY, USA, 2023. Association for Computing Machinery.

[105] R Rodríguez-Rodríguez, J Díaz, F Castro, P Ibáñez, D Chaver, V Viñals, J C Saez, M Prieto-Matias, L Piñuel, T Monreal, and J M Llabería. Reuse detector: Improving the management of stt-ram sllcs. *The Computer Journal*, 61(6):856–880, 2018.

[106] Sushil Sakhare, Manu Perumkunnil, T Huynh Bao, Siddharth Rao, Woojin Kim, Davide Crotti, Farrukh Yasin, Sebastien Couet, Johan Swerts, Shreya Kundu, et al. Enablement of stt-mram as last level cache for the high performance computing domain at the 5nm node. In *2018 IEEE International Electron Devices Meeting (IEDM)*, pages 18–3. IEEE, 2018.

[107] Soheil Salehi, Deliang Fan, and Ronald F. Demara. Survey of stt-mram cell design strategies: Taxonomy and sense amplifier tradeoffs for resiliency. *J. Emerg. Technol. Comput. Syst.*, 13(3), April 2017.

[108] Somayeh Sardashti and David A. Wood. Decoupled compressed cache: Exploiting spatial locality for energy-optimized compressed caching. In *2013 46th Ann. IEEE/ACM Int. Symp. on Microarchitecture (MICRO)*, pages 62–73, 2013.

[109] Deepika Saxena and Ashutosh Kumar Singh. Workload forecasting and resource management models based on machine learning for cloud computing environments. *arXiv preprint arXiv:2106.15112*, 2021.

[110] Stuart Schechter, Gabriel H Loh, Karin Strauss, and Doug Burger. Use ecp, not ecc, for hard failures in resistive memories. *ACM SIGARCH Computer Architecture News*, 38(3):141–152, 2010.

[111] BBC Science. The end of moore's law: what happens next? *BBC Science Focus Magazine*, 2019.

[112] Nak Hee Seong, Dong Hyuk Woo, Vijayalakshmi Srinivasan, Jude A. Rivers, and Hsien-Hsin S. Lee. Safer: Stuck-at-fault error recovery for memories. In *2010 43rd Ann. IEEE/ACM Int. Symp. on Microarchitecture*, pages 115–124, 2010.

[113] Su Myat Min Shwe, Haris Javaid, and Sri Parameswaran. Rexcache: Rapid exploration of unified last-level cache. In *2013 18th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 582–587. IEEE, 2013.

[114] Abhairaj Singh, Mahdi Zahedi, Taha Shahroodi, Mohit Gupta, Anteneh Gebregiorgis, Manu Komalan, Rajiv V Joshi, Francky Catthoor, Rajendra Bishnoi, and Said Hamdioui. Cim-based robust logic accelerator using 28 nm stt-mram characterization chip tape-out. In *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 451–454. IEEE, 2022.

[115] Y. J. Song, J. H. Lee, S. H. Han, H. C. Shin, K. H. Lee, K. Suh, D. E. Jeong, G. H. Koh, S. C. Oh, J. H. Park, S. O. Park, B. J. Bae, O. I. Kwon, K. H. Hwang, B.Y. Seo, Y.K. Lee, S. H. Hwang, D. S. Lee, Y. Ji, K.C. Park, G. T. Jeong, H. S. Hong, K. P. Lee, H. K. Kang, and E. S. Jung. Demonstration of highly manufacturable stt-mram embedded in 28nm logic. In *2018 IEEE International Electron Devices Meeting (IEDM)*, pages 18.2.1–18.2.4, 2018.

[116] David Suggs, Mahesh Subramony, and Dan Bouvier. The amd "zen 2" processor. *IEEE Micro*, 40(2):45–52, 2020.

[117] Shivam Swami and Kartik Mohanram. Reliable nonvolatile memories: Techniques and measures. *IEEE Design & Test*, 34(3):31–41, 2017.

[118] Rangharajan Venkatesan, Vivek Kozhikkottu, Charles Augustine, Arijit Raychowdhury, Kaushik Roy, and Anand Raghunathan. Tapecache: A high density, energy efficient cache based on domain wall memory. In *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design*, pages 185–190, 2012.

[119] Oreste Villa, Daniel R Johnson, Mike Oconnor, Evgeny Bolotin, David Nellans, Justin Luitjens, Nikolai Sakharnykh, Peng Wang, Paulius Micikevicius, Anthony Scudiero, et al. Scaling the power wall: a path to exascale. In *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 830–841. IEEE, 2014.

[120] Jue Wang, Xiangyu Dong, and Yuan Xie. Point and discard: a hard-error-tolerant architecture for non-volatile last level caches. In *Proceedings of the 49th Annual Design Automation Conference*, pages 253–258, 2012.

[121] 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 Int. Symp. on High Performance Computer Architecture (HPCA)*, pages 234–245. IEEE, 2013.

[122] Rujia Wang, Lei Jiang, Youtao Zhang, Linzhang Wang, and Jun Yang. Selective restore: An energy efficient read disturbance mitigation scheme for future stt-mram. In *Proceedings of the 52nd Annual Design Automation Conference*, pages 1–6, 2015.

[123] 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 Int. Symp. on High Performance Computer Architecture (HPCA)*, pages 13–24, 2014.

[124] Liqiong Wei, Juan G. Alzate, Umut Arslan, Justin Brockman, Nilanjan Das, Kevin Fischer, Tahir Ghani, Oleg Golonzka, Patrick Hentges, Rawshan Jahan, Pulkit Jain, Blake Lin, Mesut Meterelliyoz, Jim O'Donnell, Conor Puls, Pedro Quintero, Tanaya Sahu, Meenakshi Sekhar, Ajay Vangapaty, Chris Wiegand, and Fatih Hamzaoglu. 13.3 a 7mb stt-mram in 22ffl finfet technology with 4ns read sensing time at 0.9 v using write-verify-write scheme and offset-cancellation sensing technique. In *2019 IEEE Int. Solid-State Circuits Conf. (ISSCC)*, pages 214–216. IEEE, 2019.

[125] Chris Wilkerson, Hongliang Gao, Alaa R Alameldeen, Zeshan Chishti, Muhammad Khellah, and Shih-Lien Lu. Trading off cache capacity for reliability to enable low voltage operation. *ACM SIGARCH computer architecture news*, 36(3):203–214, 2008.

[126] H.-S. Philip Wong, Simone Raoux, SangBum Kim, Jiale Liang, John P. Reifenberg, Bipin Rajendran, Mehdi Asheghi, and Kenneth E. Goodson. Phase change memory. *Proceedings of the IEEE*, 98(12):2201–2227, 2010.

[127] Bi Wu, Beibei Zhang, Yuanqing Cheng, Ying Wang, Dijun Liu, and Weisheng Zhao. An adaptive thermal-aware ecc scheme for reliable stt-mram llc design. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(8):1851–1860, 2019.

[128] Wm A Wulf and Sally A McKee. Hitting the memory wall: Implications of the obvious. *ACM SIGARCH computer architecture news*, 23(1):20–24, 1995.

[129] J. Wuu, D. Weiss, C. Morganti, and M. Dreesen. The asynchronous 24mb on-chip level-3 cache for a dual-core itanium/sup /spl reg//-family processor. In *ISSCC. 2005 IEEE Int. Digest of Technical Papers. Solid-State Circuits Conf., 2005.*, pages 488–612 Vol. 1, 2005.

[130] Cong Xu, Dimin Niu, Naveen Muralimanohar, Rajeev Balasubramonian, Tao Zhang, Shimeng Yu, and Yuan Xie. Overcoming the challenges of crossbar resistive memory architectures. In *2015 IEEE 21st Int. Symp. on High Performance Computer Architecture (HPCA)*, pages 476–488. IEEE, 2015.

[131] 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, 2013.

[132] Doe Hyun Yoon and Mattan Erez. Memory mapped ecc: Low-cost error protection for last level caches. In *Proceedings of the 36th annual international symposium on Computer architecture*, pages 116–127, 2009.

[133] Doe Hyun Yoon, Naveen Muralimanohar, Jichuan Chang, Parthasarathy Ranganathan, Norman P Jouppi, and Mattan Erez. Free-p: Protecting non-volatile memory against both hard and soft errors. In *2011 IEEE 17th Int. Symp. on High Performance Computer Architecture*, pages 466–477. IEEE, 2011.

[134] Mahdi Zahedi, Muah Abu Lebdeh, Christopher Bengel, Dirk Wouters, Stephan Menzel, Manuel Le Gallo, Abu Sebastian, Stephan Wong, and Said Hamdioui. Mnemosene: Tile architecture and simulator for memristor-based computation-in-memory. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 18(3):1–24, 2022.

[135] Mahdi Zahedi, Remon van Duijnen, Stephan Wong, and Said Hamdioui. Tile architecture and hardware implementation for computation-in-memory. In *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 108–113. IEEE, 2021.

[136] Mohamed Zahran, Kursad Albayraktaroglu, and Manoj Franklin. Non-inclusion property in multi-level caches revisited. *International Journal of Computers and Their Applications*, 14(2):99, 2007.

[137] Lunkai Zhang, Brian Neely, Diana Franklin, Dmitri Strukov, Yuan Xie, and Frederic T. Chong. Mellow writes: Extending lifetime in resistive memories through selective slow write backs. In *2016 ACM/IEEE 43rd Ann. Int. Symp. on Computer Architecture (ISCA)*, pages 519–531, 2016.

[138] Chuteng Zhou, Fernando Garcia Redondo, Julian Büchel, Irem Boybat, Xavier Timoneda Comas, SR Nandakumar, Shidhartha Das, Abu Sebastian, Manuel Le Gallo, and Paul N Whatmough. Ml-hw co-design of noise-robust tinyml models and always-on analog compute-in-memory edge accelerator. *IEEE Micro*, 42(6):76–87, 2022.

[139] Ping Zhou, Bo Zhao, Jun Yang, and Youtao Zhang. A durable and energy efficient main memory using phase change memory technology. *ACM SIGARCH Computer Architecture News*, 37(3):14–23, 2009.

[140] Yangjie Zhou, Mengtian Yang, Cong Guo, Jingwen Leng, Yun Liang, Quan Chen, Minyi Guo, and Yuhao Zhu. Characterizing and demystifying the implicit convolution algorithm on commercial matrix-multiplication accelerators. In *2021 IEEE International Symposium on Workload Characterization (IISWC)*, pages 214–225. IEEE, 2021.

[141] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.