PLOS ONE

RESEARCH ARTICLE

L2C2: Last-level compressed-contents non-volatile cache and a procedure to forecast performance and lifetime

Carlos Escuin[1]*, **Pablo Ibáñez**[1], **Denis Navarro**[2], **Teresa Monreal**[3], **José M. Llabería**[3], **Víctor Viñals**[1]

1 Departamento de Informática e Ingeniería de Sistemas - Aragón Institute for Engineering Research (I3A), Universidad de Zaragoza, Zaragoza, Spain, **2** Department of Electronic Engineering and Communications, I3A, Universidad de Zaragoza, Zaragoza, Spain, **3** Departament d'Arquitectura de Computadors, Universitat Politècnica de Catalunya · BarcelonaTech (UPC), Barcelona, Spain

* escuin@unizar.es

Check for updates

OPEN ACCESS

Citation: Escuin C, Ibáñez P, Navarro D, Monreal T, Llabería JM, Viñals V (2023) L2C2: Last-level compressed-contents non-volatile cache and a procedure to forecast performance and lifetime. PLoS ONE 18(2): e0278346. https://doi.org/10.1371/journal.pone.0278346

Editor: Ashwani Kumar, Sant Longowal Institute of Engineering and Technology, INDIA

Received: June 10, 2022

Accepted: November 10, 2022

Published: February 7, 2023

Copyright: © 2023 Escuin et al. This is an open access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Data Availability Statement: The code of the forecasting procedure is available in the following link for research purposes: https://gitlab.com/uz-gaz/l2c2-forecasting.

Funding: All authors acknowledge support in part from grants (1) PID2019-105660RB-C21, PID2019-107255GB-C22, PID2019-103939RB-I00, or PDC2021-120898-I00 from Agencia Estatal de Investigación (AEI, https://www.aei.gob.es/) and Ministerio de Ciencia e Innovación (MCIN, https://www.ciencia.gob.es), (2) gaZ: T5820R research

Abstract

Several emerging non-volatile (NV) memory technologies are rising as interesting alternatives to build the Last-Level Cache (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 paper presents a novel LLC organization designed to extend the lifetime of the NV data array and a procedure to forecast in detail the capacity and performance of such an NV-LLC over its lifetime. From a methodological point of view, although different approaches are used in the literature to analyze the degradation of an NV-LLC, none of them allows to study in detail its temporal evolution. In this sense, this work 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, compression, etc.) on the temporal evolution of the indices of interest, such as the effective capacity of the NV-LLC or the system IPC. We also introduce L2C2, a LLC design intended for implementation in NV memory technology that combines fault tolerance, compression, and internal write wear leveling for the first time. 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. For instance, as a use case, we have implemented the L2C2 cache with STT-RAM technology. It has affordable hardware overheads compared to that of a baseline NV-LLC without 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 in the manufacturing process. Compared to L2C2, L2C2+6 which adds 6 bytes of redundant capacity per entry, that means 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.

group from Aragón Goverment (http://www.aragon.es), (3) EU through FEDER and NextGenerationEU/PRTR programs by the DGA-FSE, and (4) by ISCIII under Project PI21/00440.

Competing interests: The authors have declared that no competing interests exist.

1 Introduction, motivation and contributions

The goal of the cache subsystem in a shared memory multiprocessor is to reduce the number of main memory accesses. Specifically, the shared last-level cache (LLC) filters requests from the lower-level caches turning slow main memory accesses into fast LLC hits, saving main memory bandwidth, 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 6T-SRAM cells, a technology that does not scale well in terms of density and static power [1].

In the short to medium term, non-volatile memory (NVM) technologies rise as an alternative to SRAMs due to their higher density and lower static power. Among these technologies we can mention phase change (PCM) [2–4], magnetic tunnel junction (STT-RAM) [1, 5–8], or resistive (ReRAM) [9, 10].

However, write operations on most NVMs cause noticeable wear on their bitcells, making their lifetime much shorter than that of SRAMs. The simplest way to deal with an uncorrectable fault in a bitcell is to disable the memory region to which it belongs, with a size that depends on the context: a whole memory page, a cache frame or a byte. From now on, we will use the term cache *frame* to designate the set of physical bitcells of the data array holding a cache *block*, compressed or not.

In this paper we present two contributions to the design and evaluation of NV-LLCs made up with memory bitcells that wear out with writes. First, L2C2, a new fault-tolerant last-level cache organization intended for NV technologies that relies on byte disabling and data compression to increase lifetime while keeping performance. The design called L2C2+N is the endurance-scaled version of L2C2 with no more than adding N spare bytes. Second, we introduce a procedure to forecast the time evolution of effective capacity and performance, suitable for modeling either frame disabling or byte disabling with compression.

1.1 L2C2: Last-level compressed-contents NV cache

It is inherent to NVM technologies that writes deteriorate the memory bitcells. This is why NV 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, many new 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. It has been proposed, for example, to reduce the number of inserted cache blocks using some kind of filtering [11–13], or collaborating with the private levels [6]. Other techniques to reduce writes are closely tied to particular bitcell designs, supporting e.g., read-before-write [4], or early-write-termination [14, 15]. 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 [12, 16, 17].

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 [4, 18–20]. 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 bit cells, 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-RAM bit cells, 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-RAM 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 [22–24]. These techniques are orthogonal to our proposal since they deal with healthy bit cells. They could therefore be integrated into L2C2, which seeks, in a complementary way, to maintain the population of healthy bit cells 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 a mechanism that handles an error correction code (ECC), capable of detecting at least two errors and correcting one (SEC-DED), and often capable of coping with double-error correction and triple-error detection (DEC-TED). For example, Wu et al. to mitigate read disturbance errors in STT-RAM LLCs, propose to dynamically switch between SEC-DED to DEC-TED, and vice-versa, according to a temperature threshold for individual cache banks [25]. 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 [26]. 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 ECC codes 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 [27, 28]. 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 a finer control of the disable granularity, allowing the disabling of individual bytes in each frame and therefore, together with block compression, allows to increase the cache lifetime.

Alternatively, redundancy can be added outside the ECC mechanism by noting permanently failed bitcells and correcting their value [29–31]. 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 [29]. 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 the Dynamically Replicated Memory (DRM) technique to store a memory page in two partially faulty page frames [32]. Or, with a higher complexity, Jadidi et al. advocate the use of compression to harden main memory [33]. They assume a PCM memory 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 write leveling mechanism. However,

although its ideas are inspiring, this proposal has been developed to collaborate with 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 it has never been proposed to extend lifetime. For example, Choi et al. explores an adaptation of the DCC compression scheme proposed for SRAM caches [34], but applying it to embedded NVM caches [35]. Just as with 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 Error (RDE) in STT-RAM [36]. 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 [37]. The first read leaves one of the two copies unusable by the RDE, 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 [38].

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 [39]. 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.

Contributions. L2C2 is the first NV-LLC capable of tolerating byte faults in NVM bitcells. It uses data block compression and intra-frame write leveling 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 [39], by incorporating the necessary metadata to identify non-operational bytes. Besides, a SECDED mechanism is incorporated with the ability to trigger an Operating System routine that disables a byte by modifying such metadata.
- BDI compression [37]. 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 [39]. After appropriate experimentation, this option is selected. LRU-Fit is a locality-aware replacement algorithm, which selects the 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 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.

1.2 Forecasting the capacity and performance evolution of NV-LLCs

Previous work on aging and degradation of NV-LLCs often highlights the difficulty of accurately modeling aging in NV memory 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 NV main memory or cache, focuses exclusively on measuring either the reduction in the number of writes or their variability [18, 40, 41]. For instance, Wang et al. compare wear-leveling mechanisms in NV-LLCs by calculating the elapsed time from startup to the first bitcell fault [18]. 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 instructions per cycle (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 [29–33].

Ipek et al. [32] and Seong et al. [30] 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 [30] or until the memory loses all its capacity (each page is deactivated when it reaches its write limit) [32]. 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 [31], but assume that the page write rate is constant, thus expressing memory lifetime in elapsed time, rather than number of writes.

Schechter et al. [29] and Jadidi et al. [33] 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 live pages and calculate the bitcell that will fail first [29]. 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 live 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 [33]. 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, to date no procedure capable of accurately estimating the simultaneous degradation of capacity and performance over time has been proposed.

1.3 Forecasting workload behavior in cloud data centers: A seemingly similar problem

Before focusing on how to forecast capacity degradation in NV-LLCs, let's consider a problem that is also related to the interaction between workload and hardware, to assess whether its solutions are applicable. The problem is very relevant today and 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 [42, 43].

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 service level agreements (SLAs) made with customers [42, 43]. 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 [44–48].

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 [44], or from estimated resource life-cycle probabilities [47]. The second, more widespread, considers time series recorded in real data centers annotated with the relevant events [45, 46, 48].

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 [44]. The relevant workload events are generated by Cloudsim, a federated cloud data center simulator [49].

- 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 [47]. 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 [46, 48], including also its energy consumption [45]. 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 [45]. Leka et al. propose to handle the time series by chaining two neural networks, first a one-dimensional convolutional neural network (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 [46]; 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 (1D-pCNN) to learn CPU load variations at different scales [48]; 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 [50]. 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 RMSE, root-mean-square error. The validation of our model will have to be done in another way, see Section 6.1. 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 next subsection. 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.

1.4 NV-LLC memory wear out: Quantifying the problem

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

Focusing on the cache, on the one hand 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 cache write rate. On the other hand, if a certain cache set degrades more than others, it is not correct to equally distribute the write rate of the whole cache in the new associativity configuration. Let's quantify how the write rate per frame may change as the NV-LLC degrades. Fig 1 shows the average write rate per frame in a 16MB, 16-way frame-disabling NV-LLC at various aging stages (see Section 5 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* live frames in a degraded NV-LLC with 90%, 75% and 50% effective capacity, respectively.

At 90% capacity, all sets have between 16 and 7 live frames. However, when the capacity is reduced to 75%, more degraded sets appear, which only have between 6 and 1 live 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 live frames and therefore those sets experience a higher write rate, and ii) the write rate per set will be spread over fewer live frames. On the other hand, 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.

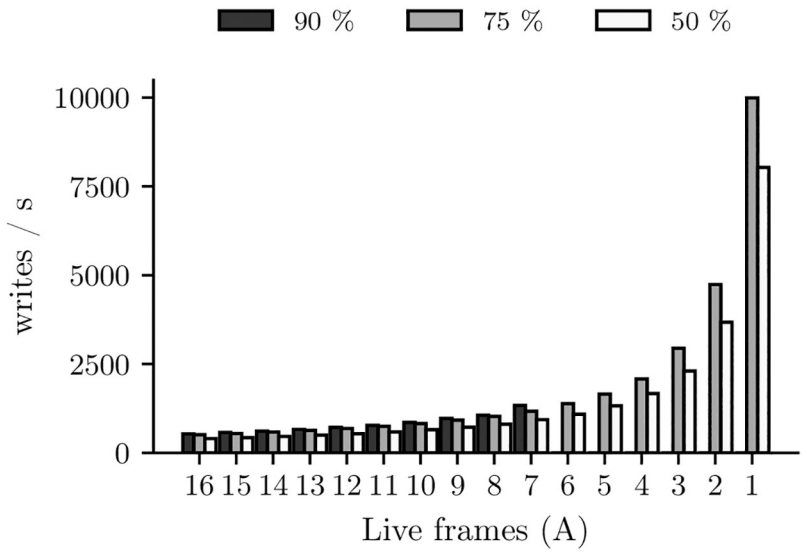

Fig 1. Average write rate per frame in sets with *A* live frames as a function of capacity (90%, 75%, and 50%).

https://doi.org/10.1371/journal.pone.0278346.g001

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 cache. 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, a single simulation cannot capture the complexity of all these interactions.

Contributions. Accurately addressing this feedback between degradation and performance loss over the lifetime of the NV-LLC is the second problem we tackle in this work. We introduce a forecasting procedure to estimate the evolution over time of any metric of interest linked to the LLC (effective capacity, miss rate, IPC, 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 cache. 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 cache 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 cache organization and the write rate statistics received. At the end of each aging prediction phase, a new cache 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 NV main memory or cache designs, although in this paper 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 in the following link: https://gitlab.com/uz-gaz/l2c2-forecasting.

The rest of the paper is organised as follows. Section 2 lays the groundwork for NV-LLCs. Section 3 describes L2C2, a byte-level fault-tolerant cache capable of handling compressed blocks, showing the storage overhead, the detailed design of the block read and write hardware, and the latency penalty incurred in the block read service. In Section 4, the forecasting procedure is conducted on systems with frame disabling and byte disabling with compression. In Section 6 we demonstrate the validity of the forecasting procedure. Section 7 evaluates the degradation of L2C2 over time and compares it to various NV-LLC configurations. Finally, Section 8 concludes this study.

2 Background

This section briefly reviews the background regarding the bitcell resilience model, data compression in the context of NV technologies, with emphasis on BDI compression, and finally, the addition of redundant capacity. The reader familiar with these concepts can skip this section without loss of continuity.

2.1 Bitcell endurance model

Writing 0 or 1 to an NVM bitcell requires to invest some energy for a time period 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 [3, 7, 51]. 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-RAM 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 (MgO) that isolates the two ferromagnetic electrodes (CoFeB): 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 antiparallel spin states, whose respective resistances are designed to be sufficiently different to encode a bit reliably [8].

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 $cv = \frac{\sigma}{\mu}$, usually between 0.2 and 0.3 [19, 29–32, 52]. 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-RAM endurance is subject to some design parameters tradeoffs such as retention time, area, power efficiency and read/write latency [53, 54]. It is therefore not surprising to find in the literature STT-RAM endurance values from 10^6 for embedded systems or IoT applications [53, 55–57] up to 10^{12} for general purpose microprocessors [18, 19, 58].

2.2 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 extends the lifetime of the main memory or cache [33, 35, 40, 41], or can be used to decrease the RDE rate [36]. Yet, compression has another benefit in the context of a byte-level fault tolerant NV cache such as L2C2: it allows cache frames with dead bytes to hold blocks if compression is high enough [33, 39]. 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 [37]. 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 (CE) is shown in Table 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.

2.3 Addition of redundant capacity

The reliability of the NV-LLC can be improved by adding redundant capacity. This can be done by using classical error detection and correction (ECC) codes or more sophisticated techniques [25, 29–32]. 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

Table 1. BDI compression encodings 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

https://doi.org/10.1371/journal.pone.0278346.t001

ECC mechanism that encodes the location of defective bitcells and assigns healthy ones to replace them [29].

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 paper.

3 Last-level compressed-contents NV cache

This section describes the basic organization of L2C2, also showing the adaptation of BDI compression, metadata layout, the details of block rearrangement and replacement, and how to add redundant capacity.

3.1 Basic organization

3.1.1 Content management between the private L1/L2 levels and the shared L2C2. Non-inclusive hierarchies have shown to be specially useful to avoid superfluous block insertions in the LLC [12]. Therefore, a non-inclusive organization is used to minimize writes in L2C2, see Fig 2. A block enters L2C2 by effect of a replacement in L2, provided that the block was not already in L2C2. In case of a write miss in L1 and L2, and a hit in L2C2, the corresponding block is brought to L1/L2 and invalidated in L2C2. Note that in this case, leaving the block in L2C2 does not make sense, because it will eventually have to be written back to L2C2 when it is evicted from L2.

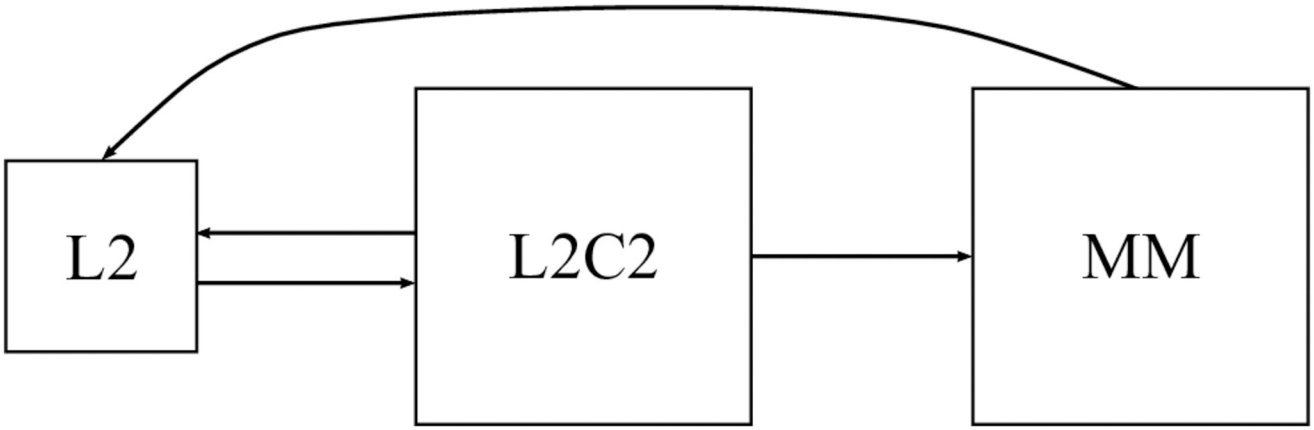

Fig 2. Block flow diagram of non-inclusive model.

https://doi.org/10.1371/journal.pone.0278346.g002

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 MRU position (lowest replacement priority), 2) a read hit in L2C2 places the block to the MRU 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 [39], 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 7.4 the two policies will be confronted.

3.1.2 Bitcell fault detection. Memory cells lose their retention capacity after a certain number of writes. It is therefore essential to handle these permanent faults without losing information. We assume a SECDED mechanism, able to correct a single-bit error and detect up to two. We assume that this ECC mechanism, upon detecting and correcting a single bit fault, triggers an Operating System exception, notifying the identity of the faulty byte [31]. Then, in order to prevent a second (uncorrectable) error from arising within the same region, the exception routine will disable the appropriate region, a whole frame using frame disabling, or a byte in L2C2. Note that ECC support is already present in many current cache designs; AMD Zen SRAM LLCs, for instance, provide DECTED [59].

3.1.3 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 [33]. Blocks are stored in the frames starting from the byte indicated by this global 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. The details and the extension of the mechanism to degraded frames can be found in Section 3.4.3.

3.2 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 [37], those marked with an * in Table 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 [39].

To quantify the importance of such low compression blocks, Fig 3 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 ≤ 37). For instance, if all

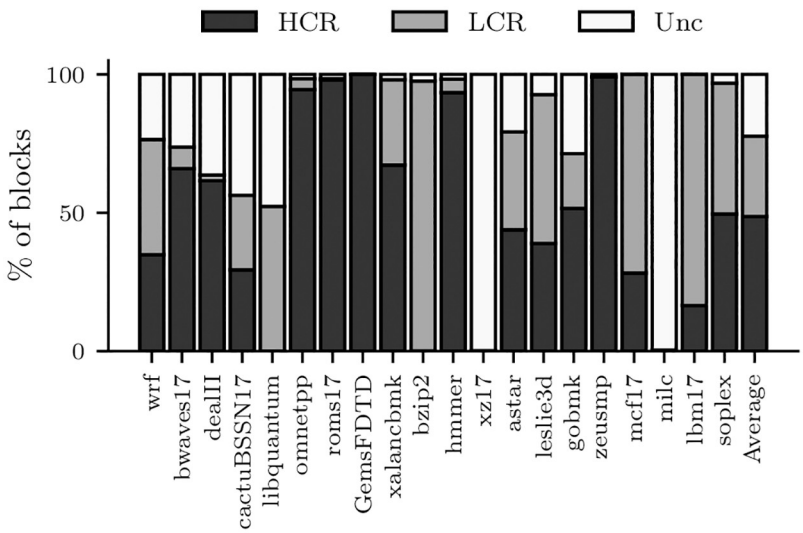

Fig 3. Block classification regarding its compression ratio for the selected SPEC CPU 2006 and 2017 applications.

https://doi.org/10.1371/journal.pone.0278346.g003

frames in an L2C2 cache have a faulty byte, and the compression mechanism does not use the low-compression ratio encodings, the chance to store 29% of the blocks would be lost.

3.3 L2C2 metadata

The tag array undergoes the most write requests as it must keep the coherence and replacement states up to date. Should these bit cells fail, the entire data frame should be deactivated. Therefore, we assume the tag array is built with 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 Fig 4).

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 the compression encoding (CE) of the data block. In addition, a fault bitmap is needed next to the data array to identify faulty bytes. This fault bitmap requires 66 bits for each frame. During the life of the frame this bitmap will experience at most 66 write requests, so it can also be implemented with NVM technology.

3.4 Block processing: Replacement and rearrangement

3.4.1 L2C2 miss, block writing. Fig 5 shows the components involved in the processing of a block B to be written in L2C2, from compression to rearrangement. In Figs 5 and 6 the shaded boxes represent what is new and/or has been modified to the Concertina proposal [39].

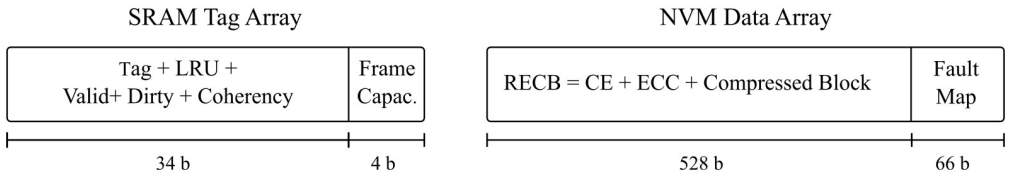

Fig 4. Layout of a frame entry in the SRAM tag and NVM data arrays.

https://doi.org/10.1371/journal.pone.0278346.g004

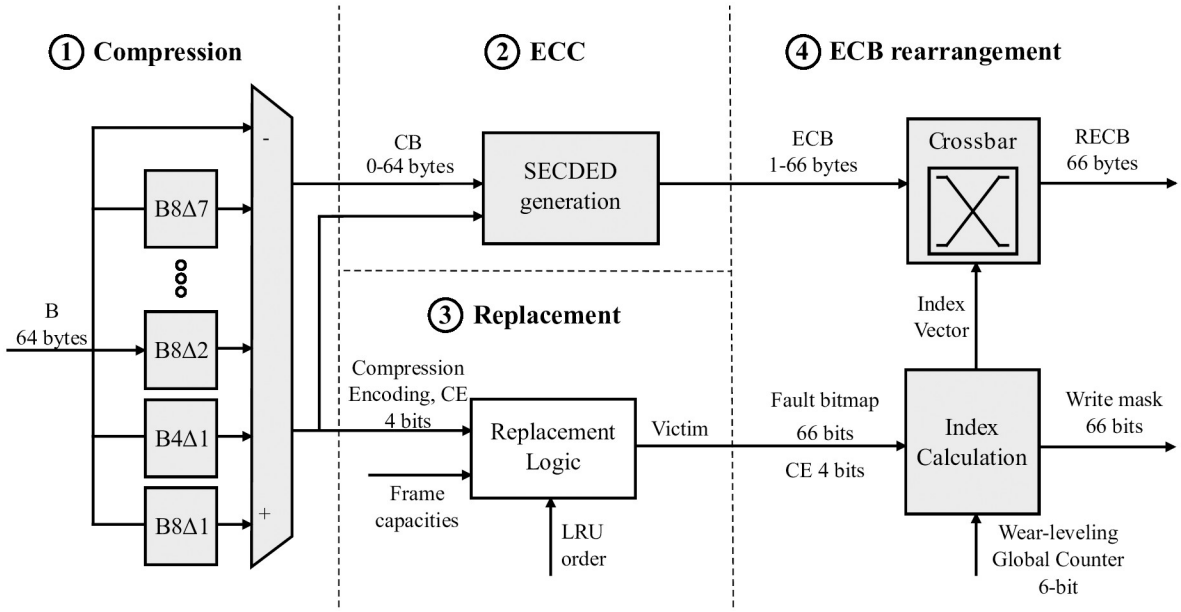

Fig 5. Flow of writing a block in L2C2 and components involved.

https://doi.org/10.1371/journal.pone.0278346.g005

First, the BDI compression units receive the block B (64 B) [① Compression]. The result of each compression unit is a) whether the block is compressible or not and, if so, b) the compressed block. As a result, the compressed block with the highest compression ratio (CB, 0–64 B) is selected and the corresponding compression encoding (CE, 4 b) is reported.

Next, the ECC bits corresponding to CB are calculated [② ECC]. The ECC mechanism selected in particular is orthogonal to our proposal. As mentioned above, we assume SECDED protection, which means an overhead between 2 and 11 bits encoded in a field of one to two bytes. We call ECB the concatenation of CB and SECDED bits, whose length ranges from 1 to 66 bytes. The length of this ECB determines the minimum capacity a cache frame must have to accommodate the block.

The replacement logic selects the victim block among the frames with the required minimum capacity [③ 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.

Every frame has an associated fault bitmap that points out the faulty bytes (66 b). This fault bitmap information is initialized to '1's indicating that all bytes in a frame are non-defective. In addition, the byte number from which to start writing the frame is reported by the Global Counter (GC, values 0–65). According to the GC and CE values and the fault bitmap, the block is rearranged for selective writing (RECB, 1–66 bytes) under a write mask (66 b) [④ Block rearrangement]. The next subsection 3) details the rearrangement logic (ECB or RECB Block rearrangement for L2C2 write or read, respectively).

3.4.2 L2C2 hit, block reading. Similarly, but in the opposite order, Fig 6 summarises the read flow of an L2C2 block. First, the block is rearranged using as input RECB, the fault bitmap and the GC value. Then, the ECC of ECB is checked, and from CB and CE the uncompressed block B is obtained and forwarded to L2/L1 [③ Decompression].

3.4.3 Rearrangement logic. The rearrangement logic is composed of two elements: Index Calculation and Crossbar. The index calculation determines the mapping from ECB bytes to

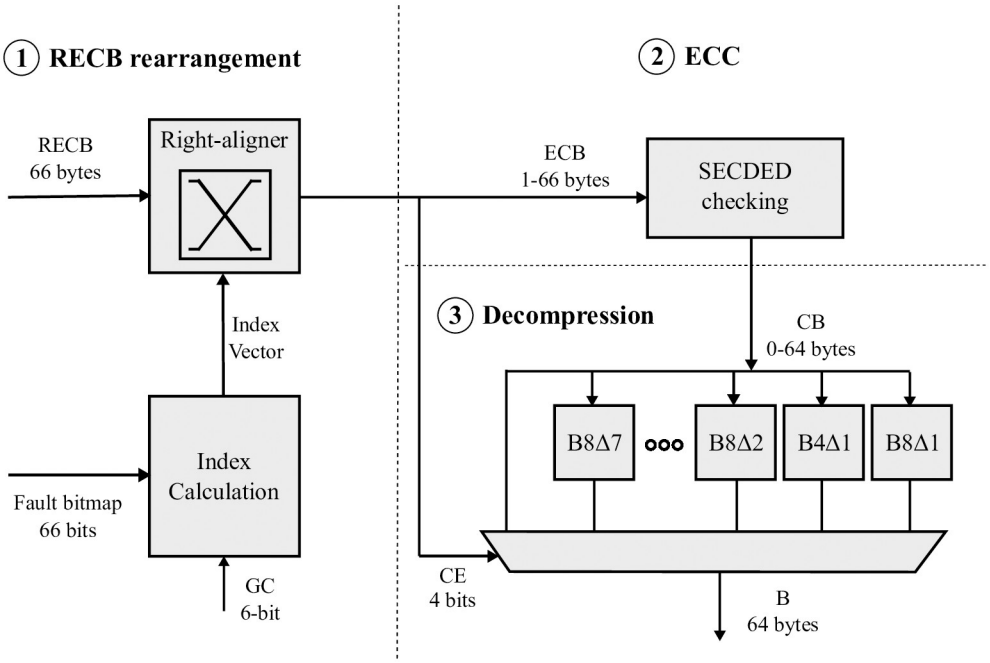

Fig 6. Flow of reading a block in L2C2 and components involved.

https://doi.org/10.1371/journal.pone.0278346.g006

RECB bytes (L2C2 write) or conversely, from RECB bytes to ECB bytes (L2C2 read). The crossbar moves bytes from the input ports to the output ports.

Fig 7 shows an example of rearranging an ECB from the fault bitmap (FM) and the GC and CE values. When writing a frame into L2C2, RECB is an ECB rearrangement consisting of a right rotation starting from the GC value and skipping the faulty bytes. Afterwards, the write is selectively performed on the bytes indicated by the computed write mask.

Algorithm 1 describes the index calculation for writing N-byte frames. It takes as inputs the fault bitmap (FM) corresponding to the destination frame and the values of the global counter (GC) and compression encoding (CE). The outputs are the write mask and the index vector I [N = *frame size*] that controls the output ports of the crossbar. For example, I [7] = 1 means that byte 1 of ECB will appear in the output port 7 of the crossbar, see Fig 7.

The first for loop (line 2) calculates indexes without considering the global counter value. That is, assuming that 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 log2 (N) adders in series. Each adder uses log2 (N) bits at most.

The two next loops (lines 5 and 6) adjust indexes considering the global counter value. Now, the iterations within each loop are independent and can be calculated in parallel, so the calculation time of the two iterations is that corresponding to two adders in series.

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

Algorithm 1: ECB → RECB Index Calculation

```
Input:
FM: N-bit vector fault bitmap
GC: global counter
```

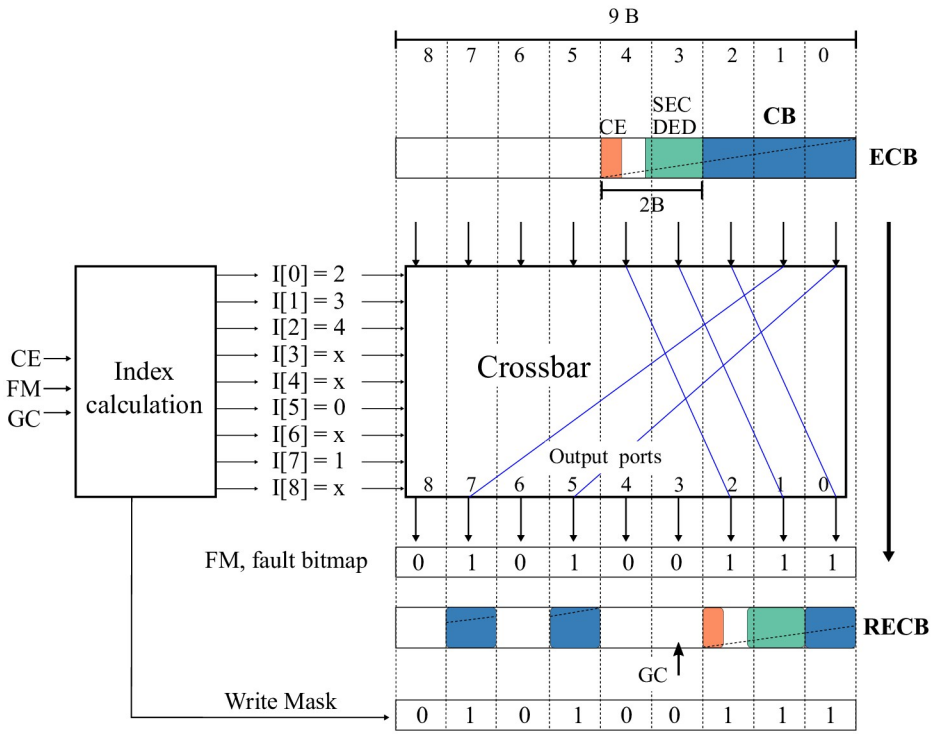

Fig 7. Example of ECB rearranging to write a 9-byte frame.

https://doi.org/10.1371/journal.pone.0278346.g007

```
  size: ECB size, computed from CE
    0 ≤ GC, size ≤ 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];
GCI = I[GC];
for i = 0; i<N; i++ do I[i] = I[i] - GCI;
for i = 0; i<GC; 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
```

The index vector is calculated 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 assumes NxN comparators of log2 (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.

VLSI implementation. To put the costs and delays of the rearrangement logic into context, we select an L2C2 built with 22nm STT-RAM technology, the largest scale of integration available in the NVsim tool [60]. Table 2 shows area, latency and power of the SRAM tag array and the STT-RAM data array, which make up the 4MB cache banks used in the experimental section.

Table 2. Hardware cost comparison.

	SRAM Tag Array 22 nm	STT-RAM Data Array 22 nm	ECB→RECB 16 nm	RECB→ECB16 nm
Area (mm^2)	0.116	0.74	0.021	0.025
Latency (ns)	0.28	2.41	0.33	0.38
Dynamic read power (mW)	0.17	6.87	-	0.49
Dynamic write power (mW)	0.16	18.64	0.61	-
Static P (mW)	109	338	0.53	0.7

https://doi.org/10.1371/journal.pone.0278346.t002

Both ECB and RECB 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 ºC 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 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 6 also provides a comparison between all the evaluated cache candidates.

3.5 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 bitmaps from 66 to 66+N bits. In addition, the rearrangement logic has to be extended to handle 66+N byte blocks, and the Global Counter has to count modulo 66+N. Without further changes, the wear-leveling logic will take care of distributing the writes among the 66+N bytes available. A frame will only start to impose performance constraints when its effective capacity falls below 66 bytes.

4 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. Without loss of generality the procedure assumes byte granularity, but extending it to other sizes is straightforward. The maximum number of writes supported by bitcells is modelled by a normal distribution.

Forecast is driven by a detailed, cycle-by-cycle simulation of a workload that can be multiprogrammed, parallel, or a mix of both execution modes. In this paper we opted for a multiprogrammed workload, but the other alternatives can be simulated in exactly the same way.

The forecasting procedure determines the live 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 each byte that fails is disabled and the remaining number of writes of those that are still live is updated.

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

4.1 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 per-byte Remaining Writes and the experienced per-byte Write Rate. 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 Fig 8A. *RW map* is initialised according to the statistical endurance model of the memory technology used, as in [19, 29–32, 52].

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 live 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 Fig 8B. 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

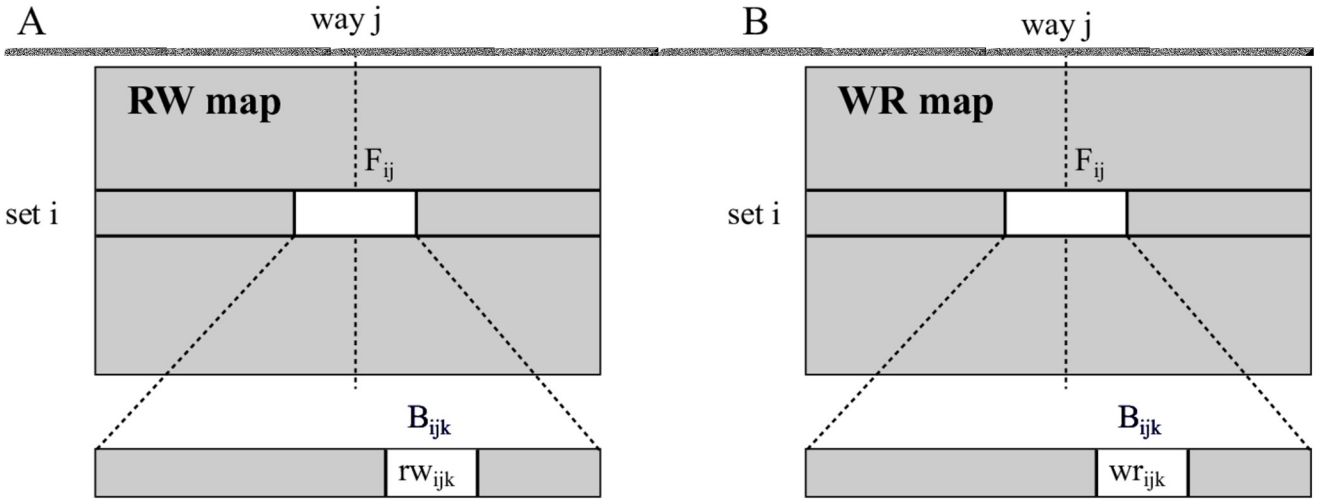

Fig 8. Per-byte Remaining Writes (A) and Write Rate (B) maps.

https://doi.org/10.1371/journal.pone.0278346.g008

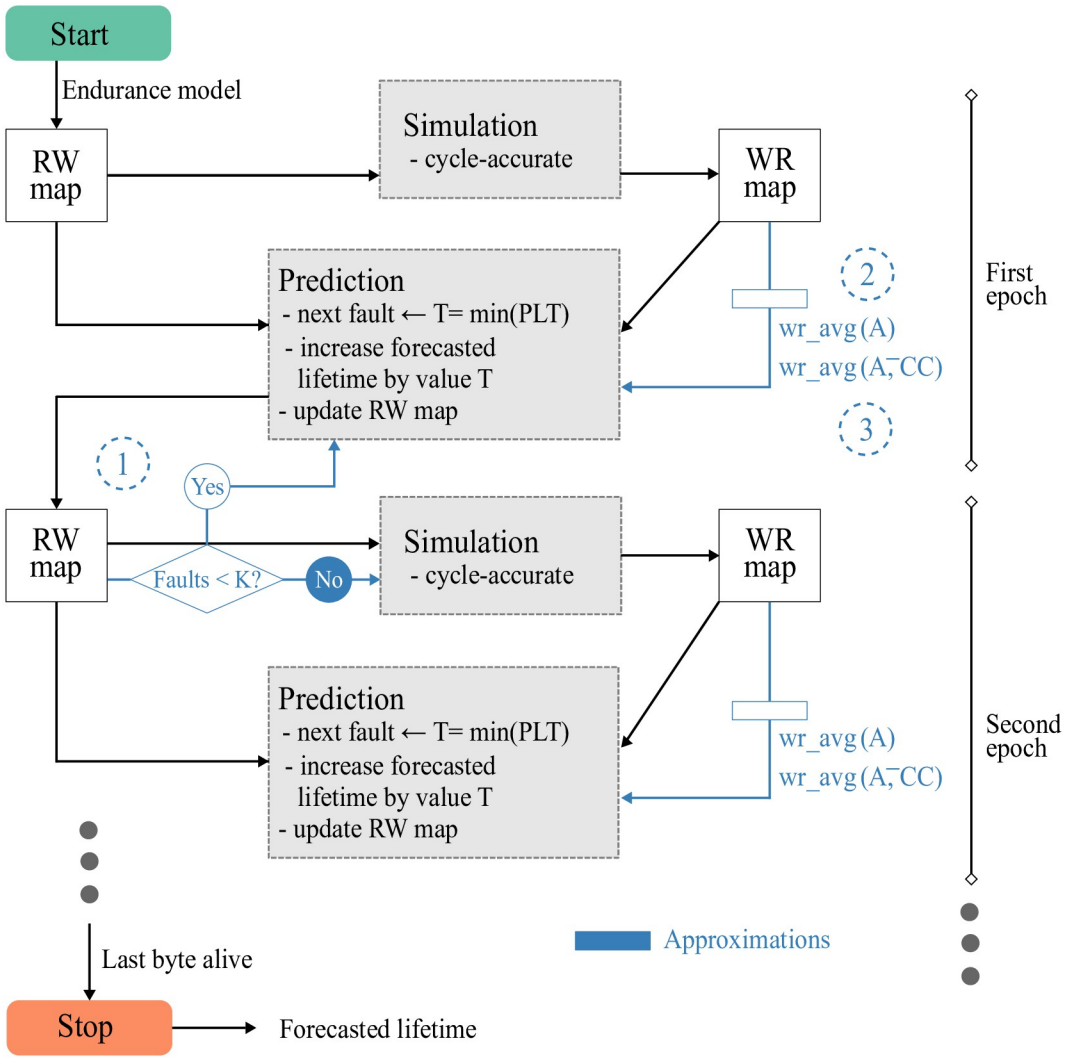

Fig 9. Forecasting procedure diagram. Basic procedure in black, approximations in blue.

https://doi.org/10.1371/journal.pone.0278346.g009

byte B_{ijk} as:

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

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

4.2 Basis of the forecasting procedure

The lifetime of an NV-LLC can be forecast using the procedure outlined with black lines in Fig 9.

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.

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. So, the simulation will 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 live 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 Fig 9.

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 Fig 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 6.3 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 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 live 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 live frames, is computed from the *WR map*; see ② in Fig 9. 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 live 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 Fig 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 live 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.

4.4 Approximate forecasting procedure for byte disabling and compression

Unlike frame disabling, in a cache with byte disabling and 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 live bytes of a frame. Consequently, during prediction, we can assume that the write rate on all live 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 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 except those of size 64 bytes (see Table 1 in page 4) and thus it is associated with CC = 58.

In this context, the prediction of write rate per byte is more complex. For example, think of 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} : $wr_avg(\bar{A}, CC)$.

More specifically,

$$wr_a vg(\bar{A}, CC) = average(wr_{ijk})$$
$$\forall\ ij\ |i)F_{ij} \in \text{set with tuple } \bar{A}$$
$$ii)F_{ij} \in \text{compression class } CC$$

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 $wr_a vg(\bar{A}, CC)$; see ③ in Fig 9.

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 $wr_a vg(\bar{A}_2, -)$ 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 $wr_a vg(\bar{A}_1, -)$ as an approximation of $wr_a vg(\bar{A}_2, -)$.

5 Methodology

Details of the multicore system modeled for the cycle-by-cycle simulation phase of each epoch are shown in Table 3. 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.

Table 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. 64 B data blocks in all levels.
L1	Private, 32 KB D, 32 KB I, 4 ways, LRU. 3-cycles load-use delay. Fetch on write miss.
L2	Private, L1-inclusive, 128 KB D, 128 KB I, 16 ways, LRU. 11-cycles load-use delay. Fetch on write miss.
STT-RAM NV-LLC	Shared, non-inclusive, 4 banks, 4MB/bank, 16 ways, LRU. Load-use delay: 30-cycles frame disabling; 32-cycles L2C2. Frames protected by SECDED. Baseline endurance: mean 10^{11} wr., cv = 0.2, 0.25, and 0.3.
Main Memory	1 memory controller, DDR4. 1 channel, 8GB/channel (1200 MHz)
NoC	Crossbar between L2C2 banks and L2s. 32 B flits.

https://doi.org/10.1371/journal.pone.0278346.t003

We use Gem5 [61] along with the Ruby memory subsystem and Garnet interconnection network. In addition, we use NVSim for the L2C2 latency estimations [60]. The workload consists of 10 mixes randomly built by SPEC CPU 2006 and 2017 benchmarks [62, 63], leaving aside applications with very little activity on the LLC [64]. Fast-forwarding is performed for the first two billion instructions and then 200M cycles are simulated in detail. Table 4 shows the applications that make up each mix along with the LLC MPKI of the mix, computed by dividing total cache misses by total number of instructions executed by all applications in the mix. Besides, the top ten memory intensive applications, in terms of accesses per kilo instruction, APKI, are superscripted.

The 10 mixes are run in the simulation phase of each epoch to obtain the *WR map* for that epoch. The write rate in each byte of the cache is calculated as the average obtained for the 10 mixes.

6 Forecasting validation, cost, and specific situations

To validate the forecasting procedure, it would be necessary to contrast its projections with data from the operation of real NVM caches 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.

6.1 Validation

As discussed in Sections 4.3 and 4.4, 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 cache degradation. Specifically, we predict how much time elapses until 50% of the cache, T_{50C}, degrades. A 50% capacity degradation is a common case study [29, 31, 33], and in our

Table 4. 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

https://doi.org/10.1371/journal.pone.0278346.t004

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 cache 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 Fig 10A and 10B 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 [29, 30, 32, 33]. 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 Fig 10 shows, in both cases but specially for compression, using a single epoch incurs in a non-negligible error.

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.

6.2 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:

$$\begin{aligned} Forecasting\ cost(f, C, E, M, Cy) &= E \cdot cost(Sim + Pred) = \\ f_1\left(f \cdot \frac{C}{K} \cdot M \cdot Cy\right) + f_2\left(f \cdot \frac{C}{K} \cdot K \cdot C\right) &= f_1(E \cdot M \cdot Cy) + f_2(f \cdot C^2) \end{aligned} \quad (1)$$

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 5 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

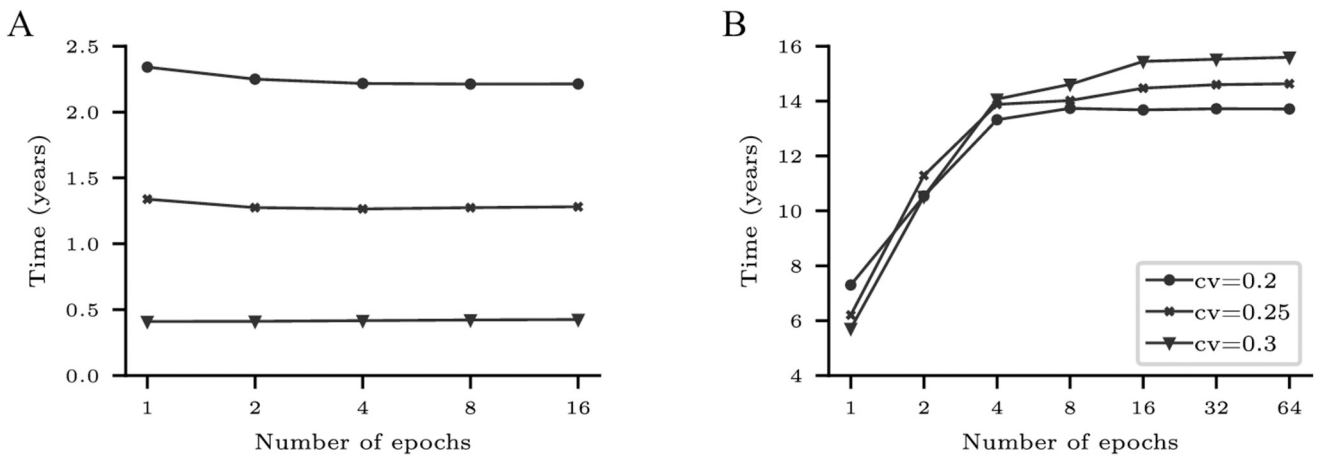

Fig 10. Forecasted T_{50C} (in years) as a function of the number of epochs for frame disabling (A) and L2C2 (B) caches. Three coefficients of variation are employed: cv = 0.2, 0.25, and 0.3.

https://doi.org/10.1371/journal.pone.0278346.g010

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.

On the other hand, as it was shown in Fig 10, 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.

6.3 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 cache 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.

Table 5. Maximum elapsed times vs. # 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

https://doi.org/10.1371/journal.pone.0278346.t005

7 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. For all tested organizations, the forecasting procedure uses 16 epochs of constant extension.

We analyze four NV-LLCs candidates, two based on frame disabling and two on byte disabling plus compression:

- **Frame disabling cache (FD)**. A bitcell failure is just handled by disabling the corresponding frame [27, 28].
- **Frame disabling cache with ECP6 (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 [29]. 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 3.
- **L2C2+6**. An L2C2 with 6 spare bytes per cache frame; see Section 3.5.

Two variations of L2C2 are also tested;

- **L2C2-NWL**. It is an L2C2 *without* the intra-frame wear-leveling mechanism. The Index Calculation circuit has less complexity; see Section 3.4.3. 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 6 shows the number of storage bits per frame of the tag and data arrays, along with the percentage increments with respect to FD.

7.1 Lifetime

Fig 11 shows forecasts of 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 7a shows T_{50C}, the time required to lose 50% of the nominal cache capacity.

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

Table 6. Frame costs in bits. Percentage overhead relative to FD.

	SRAM Tag Array		STT-RAM 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

https://doi.org/10.1371/journal.pone.0278346.t006

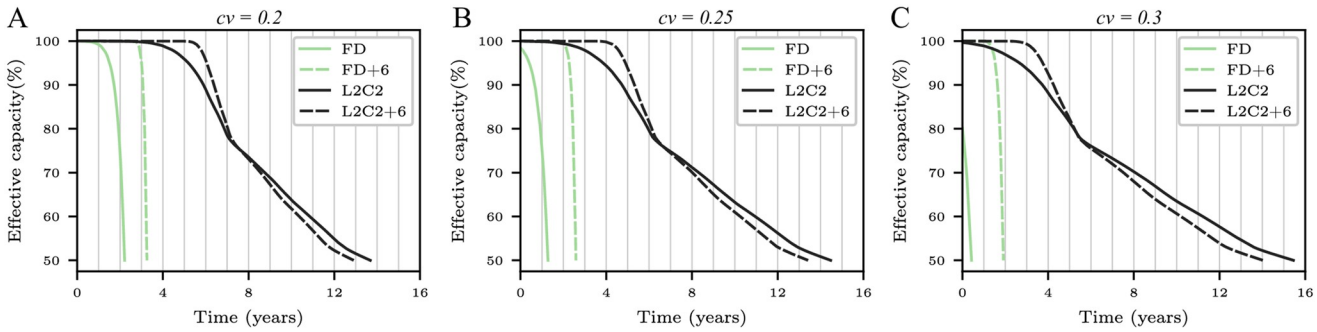

Fig 11. Effective capacity evolution over time until 50% of capacity is lost for three different cv. *cv* = 0.2 (A), *cv* = 0.25 (B), *cv* = 0.3 (C).

https://doi.org/10.1371/journal.pone.0278346.g011

bitcells; see Fig 11C. 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 7a. 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 lived 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 7b. As an example, in terms of T_{50C}, see Table 7a, 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 7. 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)		

https://doi.org/10.1371/journal.pone.0278346.t007

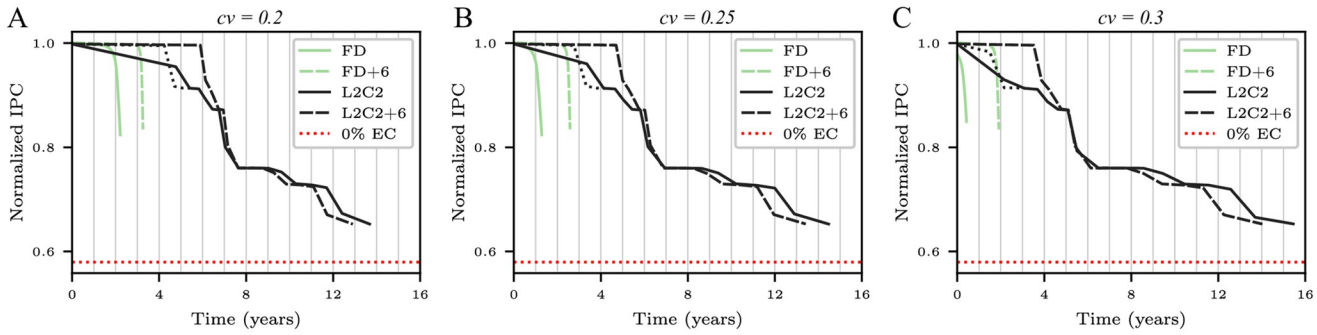

Fig 12. Normalized IPC evolution over time until 50% of capacity is lost for three different cv. $cv = 0.2$ (A), $cv = 0.25$ (B), $cv = 0.3$ (C).

https://doi.org/10.1371/journal.pone.0278346.g012

7.2 Performance

Fig 12 shows the IPC forecast over time from start-up until 50% of effective capacity is lost for the NV-LLC candidates. The IPCs have been 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.

Before going into the analysis, notice the forecasting procedure only provides IPC values after the simulation phase of each epoch, corresponding to the health state computed by the previous epoch. Intermediate IPC values within epochs are obtained by linear interpolation. However, should it be necessary to have a more precise IPC within an epoch, it is sufficient to halve the epoch extension once or several times. For instance, we observe that the first L2C2 epochs are long in forecasted time *and* produce a significant drop in IPC. To obtain more detail on the IPC loss during that period, the extension of the first two epochs has been halved, resulting in the new IPC values depicted in the dashed black lines in Fig 12.

Four observations can be highlighted from the curves.

First, after losing 50% of capacity, the IPC with frame-disabling caches is around 20% higher than that of byte-disabling. 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 capacity of all frames has been reduced and therefore some blocks cannot be stored in any frame.

Second, consistent with the effective capacity forecasts, the IPC degrades 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 7c. 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 calculate this value with the integral of the CPI curve. Since according to Belkhir et al. the average lifetime of a server is three to five years [65], we 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 7d.

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.

7.3 Intra-frame wear-leveling impact on lifetime

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.

To model the L2C2 without wear-leveling, L2C2-NWL, the write rate is not averaged across the bytes of a frame. Thus, the rate used to age a byte depends not only on the compression class of the frame it belongs to, CC, and the health state of the set, $\bar{A}$, but also on the position the byte occupies among the live bytes in the frame.

Fig 13 shows the IPC evolution until the NV-LLC loses 50% of its effective capacity for $cv = 0.2$. 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.

7.4 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 Fig 14. 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 3.1.1. 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, 78% of the total, will be allocated to this recently degraded frame; see Fig 3 in page 6. This incurring in substantial conflict misses that degrade performance.

7.5 Sensitivity analysis

To further add generality to the results presented so far, we elaborate on three aspects; see Fig 15. 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

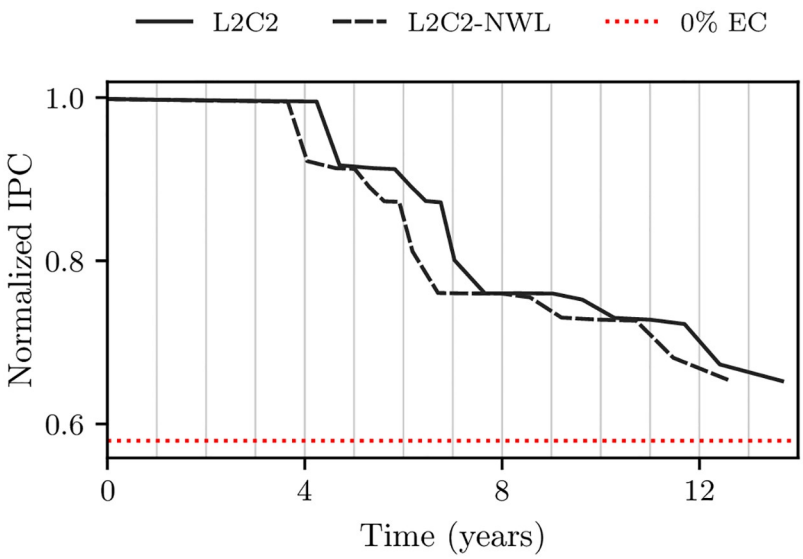

Fig 13. IPC evolution until losing 50% of capacity of an L2C2 without intra-frame wear-leveling mechanism, L2C2-NWL, for cv = 0.2.

https://doi.org/10.1371/journal.pone.0278346.g013

memory controllers. And third, the workload mixes are changed, including only the top ten memory intensive applications; see applications with superscript in the Table 4.

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 Figs 12A vs. 15A.

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 Figs 12A vs. 15B. This is an expected conclusion, which reinforces the possibility of incorporating L2C2-type caches in future generations of on-chip multiprocessors.

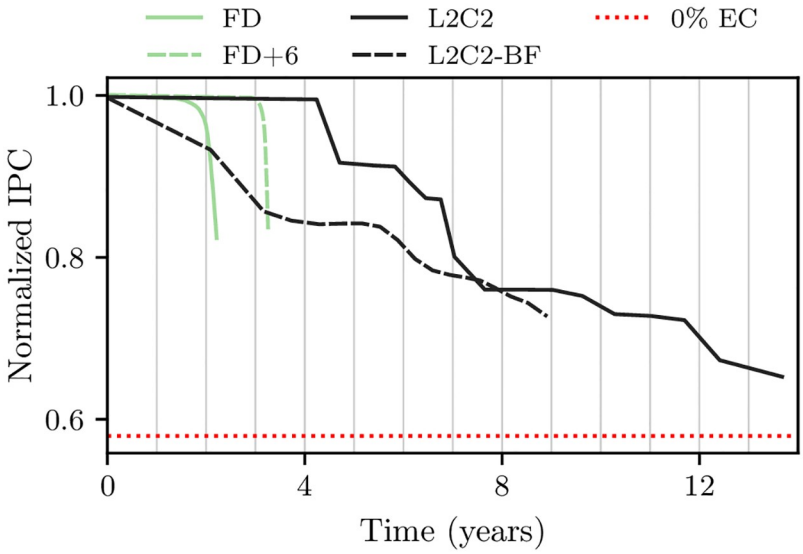

Fig 14. IPC evolution until losing 50% of capacity of an L2C2 with LRU-Best-Fit replacement policy, L2C2-BF, for *cv* = 0.2.

https://doi.org/10.1371/journal.pone.0278346.g014

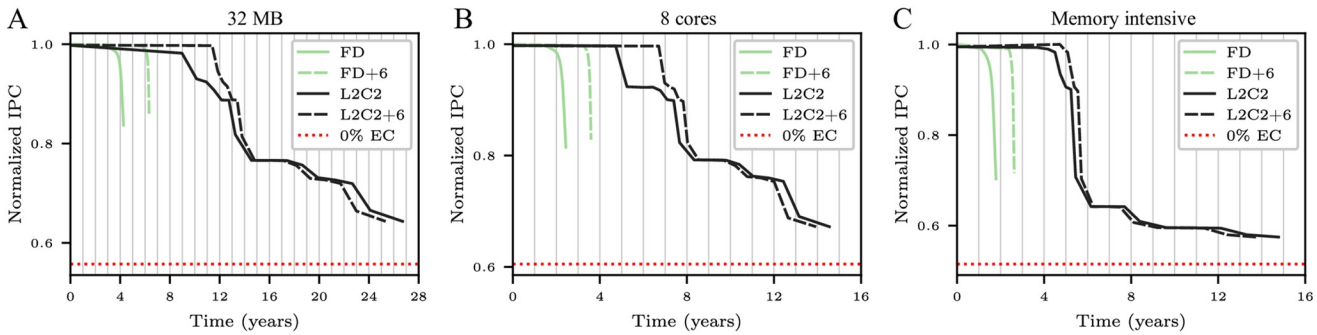

Fig 15. IPC evolution until losing 50% of capacity of FD and L2C2 for *cv* = 0.2. Doubling cache size (A), doubling the number of cores while keeping the same 4MB/core (B), and only considering the most memory-intensive programs (C).

https://doi.org/10.1371/journal.pone.0278346.g015

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 Figs 12A vs 15C. 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 both the results and the forecasting procedure itself are consistent when varying two significant dimensions, capacity and workload.

7.6 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.

All the results so far have been obtained for baseline bitcells with given endurances modelled with mean $\mu = 10^{11}$ and $cv = 0.2 - 0.3$. To obtain results concerning other bitcell endurances and/or NV-LLC latencies, of course the whole forecasting procedure can be repeated, creating new *RWmaps* 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 indices 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 Eq 8 in S1 Appendix.

Thus, from a few reference forecasts, many technology projections can be obtained. Table 8 is an example that focuses on two arbitrary, but interesting, indices: T_{90C} and T_{90P}, calculated

Table 8. T_{90C} **and** T_{90P} **for FD+6, L2C2 and L2C2+6, varying** cv **and** μ. m = months, y = years.

		Mean number of writes to fail, μ.					
cv	Cache	10^{10}		10^{11}		10^{12}	
		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	

https://doi.org/10.1371/journal.pone.0278346.t008

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 μ. 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.

These types of indices 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 [65] 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.

8 Conclusions

We have introduced L2C2, 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. 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.

On the other hand, we have developed a procedure that allows to forecast in detail the temporal evolution of such NV-LLCs. 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.

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.

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.

Supporting information

S1 Appendix. Time scaling of forecasted indexes when considering bitcells with more endurance.
(PDF)

Author Contributions

Conceptualization: Carlos Escuin, Pablo Ibáñez, Denis Navarro, Teresa Monreal, José M. Llabería, Víctor Viñals.

Data curation: Carlos Escuin, Pablo Ibáñez, Teresa Monreal, José M. Llabería, Víctor Viñals.

Formal analysis: Carlos Escuin, Pablo Ibáñez, Teresa Monreal, José M. Llabería, Víctor Viñals.

Funding acquisition: Pablo Ibáñez, Teresa Monreal, José M. Llabería, Víctor Viñals.

Investigation: Carlos Escuin, Pablo Ibáñez, Denis Navarro, Teresa Monreal, José M. Llabería, Víctor Viñals.

Methodology: Carlos Escuin, Pablo Ibáñez, Teresa Monreal, José M. Llabería, Víctor Viñals.

Project administration: Carlos Escuin, Pablo Ibáñez, Teresa Monreal, José M. Llabería, Víctor Viñals.

Resources: Carlos Escuin, Pablo Ibáñez, Denis Navarro, Teresa Monreal, José M. Llabería, Víctor Viñals.

Software: Carlos Escuin, Denis Navarro.

Supervision: Pablo Ibáñez, Teresa Monreal, José M. Llabería, Víctor Viñals.

Validation: Carlos Escuin, Pablo Ibáñez, Teresa Monreal, José M. Llabería, Víctor Viñals.

Visualization: Carlos Escuin, Pablo Ibáñez, Teresa Monreal, José M. Llabería, Víctor Viñals.

Writing – original draft: Carlos Escuin, Pablo Ibáñez, Víctor Viñals.

Writing – review & editing: Carlos Escuin, Pablo Ibáñez, Teresa Monreal, José M. Llabería, Víctor Viñals.

References

1. Sakhare S, Perumkunnil M, Bao TH, Rao S, Kim W, Crotti D, et al. Enablement of STT-MRAM as last level cache for the high performance computing domain at the 5nm node. In: 2018 IEEE Int. Electron Devices Meeting (IEDM); 2018. p. 18.3.1–18.3.4.
2. Lee BC, Ipek E, Mutlu O, Burger D. Architecting phase change memory as a scalable dram alternative. In: Proc. of the 36th annual Int. Symp. on Computer architecture; 2009. p. 2–13.

3. Qureshi MK, Gurumurthi S, Rajendran B. Phase change memory: From devices to systems. Synthesis Lectures on Computer Architecture. 2011; 6(4):1–134. https://doi.org/10.1007/978-3-031-01735-3
4. Joo Y, Niu D, Dong X, Sun G, Chang N, Xie Y. Energy-and endurance-aware design of phase change memory caches. In: 2010 Design, Automation & Test in Europe Conf. & Exhibition (DATE 2010). IEEE; 2010. p. 136–141.
5. Apalkov D, Khvalkovskiy A, Watts S, Nikitin V, Tang X, Lottis D, et al. Spin-transfer torque magnetic random access memory (STT-MRAM). ACM Journal on Emerging Technologies in Computing Systems (JETC). 2013; 9(2):1–35. https://doi.org/10.1145/2463585.2463589
6. Korgaonkar K, Bhati I, Liu H, Gaur J, Manipatruni S, Subramoney S, et al. 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). IEEE; 2018. p. 315–327.
7. Salehi S, Fan D, Demara RF. Survey of STT-MRAM Cell Design Strategies: Taxonomy and Sense Amplifier Tradeoffs for Resiliency. J Emerg Technol Comput Syst. 2017; 13(3). https://doi.org/10.1145/2997650
8. Carboni R. Characterization and Modeling of Spin-Transfer Torque (STT) Magnetic Memory for Computing Applications. In: Special Topics in Information Technology. Springer, Cham; 2021. p. 51–62.
9. Xu C, Niu D, Muralimanohar N, Balasubramonian R, Zhang T, Yu S, et al. Overcoming the challenges of crossbar resistive memory architectures. In: 2015 IEEE 21st Int. Symp. on High Performance Computer Architecture (HPCA). IEEE; 2015. p. 476–488.
10. Zhang L, Neely B, Franklin D, Strukov D, Xie Y, Chong FT. Mellow Writes: Extending Lifetime in Resistive Memories through Selective Slow Write Backs. In: 2016 ACM/IEEE 43rd Ann. Int. Symp. on Computer Architecture (ISCA); 2016. p. 519–531.
11. Rodríguez-Rodríguez R, Díaz J, Castro F, Ibáñez P, Chaver D, Viñals V, et al. Reuse detector: Improving the management of STT-RAM SLLCs. The Computer Journal. 2018; 61(6):856–880. https://doi.org/10.1093/comjnl/bxx099
12. Cheng HY, Zhao J, Sampson J, Irwin MJ, Jaleel A, Lu Y, et al. 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); 2016. p. 103–114.
13. Ahn J, Yoo S, Choi K. DASCA: Dead Write Prediction Assisted STT-RAM Cache Architecture. In: 2014 IEEE 20th Int. Symp. on High Performance Computer Architecture (HPCA); 2014. p. 25–36.
14. Zhou P, Zhao B, Yang J, Zhang Y. A durable and energy efficient main memory using phase change memory technology. ACM SIGARCH Computer Architecture News. 2009; 37(3):14–23. https://doi.org/10.1145/1555815.1555759
15. Yazdanshenas S, Pirbasti MR, Fazeli M, Patooghy A. Coding last level STT-RAM cache for high endurance and low power. IEEE Computer Architecture Letters. 2013; 13(2):73–76. https://doi.org/10.1109/L-CA.2013.8
16. Choi JH, Park GH. NVM way allocation scheme to reduce NVM writes for hybrid cache architecture in chip-multiprocessors. IEEE Trans on Parallel and Distributed Systems. 2017; 28(10):2896–2910. https://doi.org/10.1109/TPDS.2017.2689010
17. Wang Z, Jiménez DA, Xu C, Sun G, Xie Y. Adaptive placement and migration policy for an STT-RAM-based hybrid cache. In: 2014 IEEE 20th Int. Symp. on High Performance Computer Architecture (HPCA); 2014. p. 13–24.
18. Wang J, Dong X, Xie Y, Jouppi NP. 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). IEEE; 2013. p. 234–245.
19. Farbeh H, Kim H, Miremadi SG, Kim S. Floating-ECC: Dynamic repositioning of error correcting code bits for extending the lifetime of STT-RAM caches. IEEE Trans on Computers. 2016; 65(12):3661–3675. https://doi.org/10.1109/TC.2016.2557326
20. Agarwal S. LiNoVo: Longevity Enhancement of Non-Volatile Caches by Placement, Write-Restriction & Victim Caching in Chip Multi-Processors. Guwahati, India; 2020. Available from: http://gyan.iitg.ernet.in/handle/123456789/1717.
21. Cheshmikhani E, Farbeh H, Asadi H. A System-Level Framework for Analytical and Empirical Reliability Exploration of STT-MRAM Caches. IEEE Transactions on Reliability. 2020; 69(2):594–610. https://doi.org/10.1109/TR.2019.2923258
22. Cheshmikhani E, Farbeh H, Asadi H. Enhancing Reliability of STT-MRAM Caches by Eliminating Read Disturbance Accumulation. In: 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE); 2019. p. 854–859.
23. Cheshmikhani E, Farbeh H, Asadi H. 3RSeT: Read Disturbance Rate Reduction in STT-MRAM Caches by Selective Tag Comparison. IEEE Transactions on Computers. 2022; 71(6):1305–1319.

24. Cheshmikhani E, Farbeh H, Miremadi SG, Asadi H. TA-LRW: A Replacement Policy for Error Rate Reduction in STT-MRAM Caches. IEEE Transactions on Computers. 2019; 68(3):455–470. https://doi.org/10.1109/TC.2018.2875439

25. Wu B, Zhang B, Cheng Y, Wang Y, Liu D, Zhao W. An Adaptive Thermal-Aware ECC Scheme for Reliable STT-MRAM LLC Design. IEEE Transactions on Very Large Scale Integration (VLSI) Systems. 2019; 27(8):1851–1860. https://doi.org/10.1109/TVLSI.2019.2913207

26. Kim J, Hardavellas N, Mai K, Falsafi B, Hoe J. Multi-bit Error Tolerant Caches Using Two-Dimensional Error Coding. In: 40th Ann. IEEE/ACM Int. Symp. on Microarchitecture (MICRO 2007); 2007. p. 197–209.

27. Chang J, Huang M, Shoemaker J, Benoit J, Chen SL, Chen W, et al. 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. 2007; 42(4):846–852. https://doi.org/10.1109/JSSC.2007.892185

28. Wuu J, Weiss D, Morganti C, Dreesen M. 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.; 2005. p. 488–612 Vol. 1.

29. Schechter S, Loh GH, Strauss K, Burger D. Use ECP, not ECC, for hard failures in resistive memories. ACM SIGARCH Computer Architecture News. 2010; 38(3):141–152. https://doi.org/10.1145/1816038.1815980

30. Seong NH, Woo DH, Srinivasan V, Rivers JA, Lee HHS. SAFER: Stuck-At-Fault Error Recovery for Memories. In: 2010 43rd Ann. IEEE/ACM Int. Symp. on Microarchitecture; 2010. p. 115–124.

31. Yoon DH, Muralimanohar N, Chang J, Ranganathan P, Jouppi NP, Erez M. FREE-p: Protecting non-volatile memory against both hard and soft errors. In: 2011 IEEE 17th Int. Symp. on High Performance Computer Architecture. IEEE; 2011. p. 466–477.

32. Ipek E, Condit J, Nightingale EB, Burger D, Moscibroda T. Dynamically replicated memory: building reliable systems from nanoscale resistive memories. ACM Sigplan Notices. 2010; 45(3):3–14. https://doi.org/10.1145/1735971.1736023

33. Jadidi A, Arjomand M, Tavana MK, Kaeli DR, Kandemir MT, Das CR. 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); 2017. p. 85–96.

34. Sardashti S, Wood DA. Decoupled compressed cache: Exploiting spatial locality for energy-optimized compressed caching. In: 2013 46th Ann. IEEE/ACM Int. Symp. on Microarchitecture (MICRO); 2013. p. 62–73.

35. Choi JH, Kwak JW, Jhang ST, Jhon CS. Adaptive cache compression for non-volatile memories in embedded system. In: Proc. of the 2014 Conf. on Research in Adaptive and Convergent Systems; 2014. p. 52–57.

36. Mittal S. Mitigating read disturbance errors in STT-RAM caches by using data compression. In: Nanoelectronics: Devices, Circuits and Systems. Elsevier; 2019. p. 133–152.

37. Pekhimenko G, Seshadri V, Mutlu O, Kozuch MA, Gibbons PB, Mowry TC. Base-delta-immediate compression: Practical data compression for on-chip caches. In: 2012 21st Int. Conf. on Parallel Architectures and Compilation Techniques (PACT). IEEE; 2012. p. 377–388.

38. Wang R, Jiang L, Zhang Y, Wang L, Yang J. Selective restore: An energy efficient read disturbance mitigation scheme for future STT-MRAM. In: Proceedings of the 52nd Annual Design Automation Conference; 2015. p. 1–6.

39. Ferrerón A, Suárez-Grácia D, Alastruey-Benedé J, Monreal-Arnal T, Ibáñez P. Concertina: Squeezing in cache content to operate at near-threshold voltage. IEEE Trans on Computers. 2015; 65(3):755–769.

40. Dgien DB, Palangappa PM, Hunter NA, Li J, Mohanram K. Compression architecture for bit-write reduction in non-volatile memory technologies. In: 2014 IEEE/ACM Int. Symp. on Nanoscale Architectures (NANOARCH). IEEE; 2014. p. 51–56.

41. Palangappa PM, Mohanram K. CASTLE: compression architecture for secure low latency, low energy, high endurance NVMs. In: 2018 55th ACM/ESDA/IEEE Design Automation Conf. (DAC). IEEE; 2018. p. 1–6.

42. Dong H, Munir A, Tout H, Ganjali Y. Next-Generation Data Center Network Enabled by Machine Learning: Review, Challenges, and Opportunities. IEEE Access. 2021; 9:136459–136475. https://doi.org/10.1109/ACCESS.2021.3117763

43. Saxena D, Singh AK. Workload forecasting and resource management models based on machine learning for cloud computing environments. arXiv preprint arXiv:210615112. 2021;.

44. Bouaouda A, Afdel K, Abounacer R. 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); 2022. p. 150–157.

45. Khan T, Tian W, Ilager S, Buyya R. Workload forecasting and energy state estimation in cloud data centres: ML-centric approach. Future Generation Computer Systems. 2022; 128:320–332. https://doi.org/10.1016/j.future.2021.10.019

46. Leka HL, Fengli Z, Kenea AT, Tegene AT, Atandoh P, Hundera NW. 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); 2021. p. 474–478.

47. Li K. Profit Maximization in a Federated Cloud by Optimal Workload Management and Server Speed Setting. IEEE Transactions on Sustainable Computing. 2021; p. 1–1.

48. Patel E, Kushwaha DS. A hybrid CNN-LSTM model for predicting server load in cloud computing. The Journal of Supercomputing. 2022; 78(8):1–30. https://doi.org/10.1007/s11227-021-04234-0

49. Calheiros RN, Ranjan R, Beloglazov A, De Rose CA, Buyya R. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and experience. 2011; 41(1):23–50.

50. Makridakis S, Spiliotis E, Assimakopoulos V. Statistical and Machine Learning forecasting methods: Concerns and ways forward. PloS one. 2018; 13(3):e0194889. https://doi.org/10.1371/journal.pone.0194889 PMID: 29584784

51. Mittal S, Vetter JS. Reliability Tradeoffs in Design of Volatile and Nonvolatile Caches. Journal of Circuits, Systems and Computers. 2016; 25(11):1650139. https://doi.org/10.1142/S0218126616501395

52. Cintra M, Linkewitsch N. 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; 2013. p. 217–228.

53. Golonzka O, Alzate JG, Arslan U, Bohr M, Bai P, Brockman J, et al. MRAM as embedded non-volatile memory solution for 22FFL FinFET technology. In: 2018 IEEE Int. Electron Devices Meeting (IEDM). IEEE; 2018. p. 18–1.

54. Natsui M, Tamakoshi A, Honjo H, Watanabe T, Nasuno T, Zhang C, et al. 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;.

55. Chih YD, Shih YC, Lee CF, Chang YA, Lee PH, Lin HJ, et al. 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). IEEE; 2020. p. 222–224.

56. Lee YK, Song Y, Kim J, Oh S, Bae BJ, Lee S, et al. Embedded STT-MRAM in 28-nm FDSOI logic process for industrial MCU/IoT application. In: 2018 IEEE Symp. on VLSI Technology. IEEE; 2018. p. 181–182.

57. Wei L, Alzate JG, Arslan U, Brockman J, Das N, Fischer K, et al. 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). IEEE; 2019. p. 214–216.

58. Huai Y. Spin-transfer torque MRAM (STT-MRAM): Challenges and prospects. AAPPS bulletin. 2008; 18(6):33–40.

59. Suggs D, Subramony M, Bouvier D. The AMD "Zen 2" Processor. IEEE Micro. 2020; 40(2):45–52. https://doi.org/10.1109/MM.2020.2974217

60. Dong X, Xu C, Xie Y, Jouppi NP. Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. IEEE Trans on Computer-Aided Design of Integrated Circuits and Systems. 2012; 31(7):994–1007. https://doi.org/10.1109/TCAD.2012.2185930

61. Lowe-Power J, Ahmad AM, Akram A, Alian M, Amslinger R, Andreozzi M, et al. The gem5 simulator: Version 20.0+. arXiv preprint arXiv:200703152. 2020;.

62. Henning JL. SPEC CPU2006 benchmark descriptions. ACM SIGARCH Computer Architecture News. 2006; 34(4):1–17. https://doi.org/10.1145/1186736.1186737

63. Bucek J, Lange KD, v Kistowski J. SPEC CPU2017: Next-Generation Compute Benchmark. In: Companion of the 2018 ACM/SPEC Int. Conf. on Performance Engineering. ICPE'18. New York, NY, USA: Association for Computing Machinery; 2018. p. 41–42.

64. Navarro-Torres A, Alastruey-Benedé J, Ibáñez-Marín P, Viñals-Yúfera V. Memory hierarchy characterization of SPEC CPU2006 and SPEC CPU2017 on the Intel Xeon Skylake-SP. Plos one. 2019; 14(8): e0220135. https://doi.org/10.1371/journal.pone.0220135 PMID: 31369592

65. Belkhir L, Elmeligi A. Assessing ICT global emissions footprint: Trends to 2040 & recommendations. Journal of cleaner production. 2018; 177:448–463. https://doi.org/10.1016/j.jclepro.2017.12.239