ECC-Map: A Resilient Wear-Leveled Memory-Device Architecture with Low Mapping Overhead

Natan Peled
Viterbi Department of ECE
Technion - Israel Institute of Technology
Haifa, Israel
natanpeled@campus.technion.ac.il

Yuval Cassuto
Viterbi Department of ECE
Technion - Israel Institute of Technology
Haifa, Israel
ycassuto@ee.technion.ac.il

ABSTRACT

New non-volatile memory technologies show great promise for extending the memory hierarchy, but have limited endurance that needs to be mitigated toward their reliable use closer to the processor. Wear leveling is a common technique for prolonging the life of endurance-limited memory, where existing wear-leveling approaches either employ costly full-indirection mapping between logical and physical addresses, or choose simple mappings that cannot cope with extremely unbalanced write workloads. In this work, we propose ECC-Map, a new wear-leveling device architecture that can level even the most unbalanced and adversarial workloads, while enjoying low mapping complexity compared to full indirection. Its key idea is using a family of efficiently computable mapping functions allowing to selectively remap heavily written addresses, while controlling the mapping costs by limiting the number of functions used at any given time. ECC-Map is evaluated on common synthetic workloads, and is shown to significantly outperform existing wear-leveling architectures. The advantage of ECC-Map grows with the device's size-to-endurance ratio, a parameter that is expected to grow in the scaling trend of growing capacities and shrinking reliabilities.

CCS CONCEPTS

• **Hardware** → **Non-volatile memory**; *Hardware reliability*; *Analysis and design of emerging devices and systems*; *Memory and dense storage*; • **Information systems** → *Storage class memory*.

KEYWORDS

Non-volatile memory, persistent memories, wear-leveling, error-correcting codes.

1 INTRODUCTION

Computing systems are challenged by the growing memory demands of data-intensive applications. These demands grow faster than the scaling of DRAM, the principal main-memory technology [13]. Thus, new memories, called persistent non-volatile (NV) memories are being designed and deployed in emerging computing architectures. These memories have lower cost than DRAM memories, and faster access than non-volatile storage technologies such as NAND Flash. The design challenge of NV memories lies in their "sandwich" status: having both performance expectations of main memory and cost expectations of backing storage. NV memories already exist in a variety of technologies such as PCM, RERAM, MRAM [1, 9, 20], and others, but it is expected that scaled-up versions of these (or other) technologies will find an even more dominant role in the future. In those systems, it is expected that commercial NV memory devices will have large capacities, small data units (lines) for fast access, and limited endurance due to low cost/high density.

As NV memories take upon the demanding workloads of data-intensive applications, concern is raised about their limited endurance. This concern is not new: early phase-change memory (PCM) based main-memory architectures already addressed their limited endurance by devising wear-leveling mechanisms. However, the existing solutions are not sufficient to mitigate the problem in the scaling trend of growing memory capacities and shrinking endurances (due to increased density). In fact, the wear-leveling problem becomes dramatically more challenging as the capacity grows or as the endurance decreases (and doubly so if both happen simultaneously). Consequently, we revisit the wear-leveling problem in this work, and propose a new *device architecture* we call *ECC-Map* to support flexible wear leveling with low implementation costs.

Even though Flash devices suffer from a similar endurance problem [21], existing solutions for Flash devices do not fit persistent NV memories. Writing in Flash devices is split between program and erase operations [11], where each works at a different granularity. The effect of this is that individual lines ("pages" in the Flash terminology) cannot be re-written in-place, which necessitates out-of-place writing and frequent data movements (garbage collection) [17]. For that purpose, most Flash storage products implement a full-indirection translation layer in their controllers. Full indirection using mapping tables makes the wear-leveling problem easier to handle, but with the high cost of using large tables in expensive controller memory. In the case of persistent NV memories, for example in PCM, the device cells do not have to be erased before writing to them. In addition, the access granularity is typically finer than in Flash devices. Those properties obviate the need for garbage collection, allowing to use more economical mechanisms for wear leveling compared to full indirection tables.

Toward a clear presentation of the problem and our ECC-Map architecture, we use a simple memory-device model. The device is accessed by a host computing system through a read/write interface spanning a linear range of logical addresses, and it employs an internal controller for managing the read/write of the physical memory. The device-host read/write interface uses data units we call *lines*. From the host side, a line is addressed by its *logical line address (LLA)*, and internally a line is stored in a *physical line address (PLA)*. The device has N PLAs in total, and every PLA can be written at most w_{max} times in the device lifetime. The basic need that the device architecture needs to fulfil is *mapping flexibility* between

LLAs and PLAs, such that even if LLA write loads are extremely unbalanced, no PLA will exceed its endurance limit w_{max} prematurely. In addition to mapping flexibility, the architecture needs to specify the *wear-leveling algorithms* governing the evolution of this mapping in the device lifetime, and the internal data movements.

A well-known wear-leveling architecture, called *Start-Gap (SG)* [14], uses a very economical mapping structure and allowing basic address remappings called gap movements. The gap is a spare PLA, and its movement is done by writing into it the LLA residing in the adjacent PLA. While SG has demonstrated good wear-leveling performance in some natural workloads, in extremely unbalanced workloads its performance is not satisfactory. In particular, when w_{max} is not orders of magnitudes larger than N, SG fails to adequately level an adversarial workload that continuously writes to a single LLA, which we call the *1-LLA workload* in this paper. This is a major potential impediment in practice given the earlier stated trend of growing N and shrinking w_{max}. Some mitigations have been proposed for this undesired behavior, but none of them fully solve the problem. Dividing the device to regions is a common useful proposition made in [14, 15, 26] (and others); but with the blessing of breaking N into small "mini-devices" comes the curse (and complexity) of simultaneously managing many such mini-devices. Other proposed approaches to deal with extremely unbalanced workloads have been to cache those writes in an unlimited-endurance media, or better yet, to block the writes from the device by caching them "elsewhere". While these may work in specific system settings, a stand-alone persistent-memory device can assume neither of the two, and must guarantee adequate wear leveling on its own.

The new device architecture we present in this paper aims to solve the wear-leveling problem by enhancing the flexibility of the LLA-PLA mapping, while keeping the costs associated with this new mapping small and controlled. The fundamental problem of wear-leveling mapping architectures is that they must be able to map frequently-written LLAs flexibly across the PLA space, and tracking a flexible workload-dependent mapping costs memory and processing resources. If any LLA can be heavily written, resulting in its repeated remapping within the entire PLA space, it appears intuitively necessary for the device to keep for each LLA its current full PLA address. Storing and maintaining such a mapping entails high cost (memory space) and complexity (persisting and/or wear-leveling meta-data), which we would like to avoid for commercial viability. Fortunately, contradicting this intuition, we show in the sequel a mapping architecture that is able to relocate any LLA across the entire PLA space, without maintaining costly LLA to PLA mapping.

The main idea of ECC-Map is to use *a family of mapping functions* to maintain the LLA-PLA mapping. A large family of functions gives more flexibility than the simple functions used in prior work, and at the same time more efficiency than offered by a mapping table in memory. Around these mapping functions we design the entire mapping architecture and its algorithms, which we summarize in the following by stating ECC-Map's main ingredients.

(1) **A family of** efficiently computable **mapping functions** with properties allowing effective reclaiming of unused wear. Each member of the family is defined by an integer *mapping index.*
(2) **A sliding window** bounding the range of mapping indices used throughout the device at a given time. The window size controls the mapping complexity.
(3) **Selective remapping** of specific logical addresses from their current physical locations to a new location determined by a subsequent mapping index.
(4) **A remapping trigger** invoked when a physical location reaches a designated wear threshold based on either a write-count estimate or reliability estimate.
(5) **Mapping-index randomization** to prevent an adversary from tracking mapping pairs and generating harmful adaptive workloads.

Ingredients 1-4 are new, to the best of our knowledge, while ingredient 5 is a commonly used security measure.

We provide the details of the ECC-Map architecture in Section 2, and in Section 3 we present the performance evaluation of ECC-Map on a device discrete-event simulation. The results show that high device utilizations can be reached even for N/w_{max} ratios that fail prior wear-leveling architectures. The promising results of the modeled device in the simulation environment motivate the future implementation of ECC-Map in a hardware device within a working computing system. For that, in Section 4 we discuss finer implementation details toward a more practical realization of the architecture in hardware.

2 THE MAPPING ARCHITECTURE

It is imperative upon a memory device to maximize its ability to serve host writes before reaching its physical endurance limits. In this work, we assume an endurance model whereby each PLA is limited to w_{max} total physical writes, and once exceeded, it cannot be used anymore for read or write. To guarantee full usability, we define the device's *lifetime* as the time until *any PLA* exceeds w_{max} writes. Hence, the key performance objective in this work is to maximize the total number of host writes served by the device in its lifetime. Let N be the number of PLAs in the device, that is, its physical capacity in units of lines. Then $w_{max} \cdot N$ is a fundamental upper bound on the number of host writes served within the device lifetime. With respect to this fundamental bound, we define the writing *utilization* as

$$Utilization = \frac{\#Host\ writes}{w_{max} \cdot N}, \quad (1)$$

which corresponds to a specific write workload served by the device. The utilization is a number between 0 and 1, the higher the better. Note that the utilization measure accounts for the *full storage cost* N, even in the case of over-provisioned physical storage $N > K$, where K is the number of LLAs. Furthermore, the numerator counts *host writes* and not physical writes (the latter include internal writes by the mapping layer), and thus (1) captures the true utility offered by the device to the customer. In contrast, some prior works (e.g., [14]) use performance measures that count the total number of physical writes (including internal writes), and are thus not valid in cases of significant write amplification. Later in the paper we evaluate the utilization for several important workloads, focusing primarily on notoriously challenging workloads.

Improving the utilization by wear leveling is made possible by implementing a *mapping layer* that spreads the uneven LLA access more evenly across the PLA space. The mapping layer maintains a dynamic *mapping function* between the K LLAs and the N PLAs; in general $N \geq K$, and we define $\rho = (N-K)/N \geq 0$ as the *spare factor* of the device. At any point in time, the mapping function needs to be *injective*, that is, not mapping multiple LLAs to the same PLA. The function needs *not* be surjective, that is, not all PLAs need to map to LLAs. The simplest but most costly implementation of the mapping function is by a mapping table having an entry for each LLA storing its mapped PLA. A mapping table can wear level effectively, but its hardware and maintenance costs are prohibitive. Much more efficient mapping layers use a global efficiently computable function: $PLA = f(LLA)$, where $f(\cdot)$ changes in time, and storing its specification can be done with little memory. The key of the proposed mapping architecture of this work is to extend this from a single function $f(\cdot)$ to a *family of functions* $\{f_i(\cdot)\}$, allowing different LLAs to be mapped using different mapping indices i. The combined mapping function, which maps every LLA to the PLA output by the function $f_i(\cdot)$ designated for it, needs to be an injective mapping at every given time. Adding the index to the mapping function improves its flexibility to level the wear, while bounding the mapping cost is possible by limiting the range of indices used throughout the LLA space at any given time.

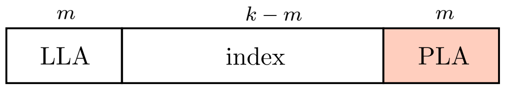
2.1 Implementation of the Mapping Functions

To implement the family of functions, we use encoding functions of cyclic error-correcting codes (ECC), used elsewhere for error correction and detection, including as cyclic redundancy check (CRC) [12] codes. We choose these functions for the several advantages they offer: 1) efficient hardware implementation, 2) simple reverse mapping, and 3) spreading an LLA mapping across the entire PLA space (as we detail later). The third feature is critical for obtaining high utilization in adversarial workloads, and is in general not satisfied by alternative options such as cryptographic pseudo-random permutations. Common cryptographic functions also require significantly higher computation load relative to cyclic ECC encoding.

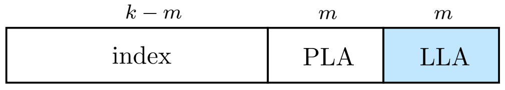
Assuming N (the number of PLAs) is an integer power of 2, we define an integer parameter $m = \log_2 N$. For the function family we take a binary cyclic ECC with parameters $[n, k]$, where n is the codeword length and k is the number of information bits input to the encoder. $r = n - k$ is the redundancy of the code, and the code is specified by a binary generator polynomial of degree r. A convenient source for such codes is the family of primitive BCH codes that exist for a rich variety of $[n, k]$ combinations; some sample generator polynomials of BCH codes can be found in [7]. We choose a code with $r = m$ and $k \geq 2m$. The input to the encoder is the binary vector $[LLA|i]$, where | represents concatenation and i is the mapping index. LLA is represented as an m-bit vector and i as a $(k-m)$-bit vector, both using the standard binary representation. The encoding is depicted in Figure 1a. The output of the encoder is an $r = m$-bit representation of the output PLA. Using the encoder as specified, we get the following.

Property 1. *$f_i(\cdot)$ is an injective function for every index i.*

Property 1 is proven by contradiction: assume there are two different LLAs mapping to the same PLA for the same index. Then by subtracting the corresponding two codewords (modulo 2), we



(a) Forward mapping.



(b) Inverse mapping.

Figure 1: (a) Forward mapping. The input LLA and the function index (in white) comprise the input to the ECC encoder. The encoder's output (shaded orange) gives the PLA resulting from the mapping. (b) Inverse mapping. PLA and LLA exchange roles: PLA is now part of the input (in white) and LLA is the output (shaded blue). Since this layout is a cyclic shift from the forward mapping, the exact same encoder function can be used.

get (from linearity) a third codeword all of whose non-zeros are confined to m or less consecutive coordinates, which contradicts a known property of cyclic codes with redundancy $r = m$.

The inverse mapping (from PLA and index to LLA) is shown in Figure 1b: the input is $[i|PLA]$ and the output is LLA. From the cyclic property of the code and the fact that Figure 1b is obtained from Figure 1a by a cyclic shift of m positions to the left, the inverse mapping can use the same encoding function used by the forward mapping, but note the required reordering of the input arguments $[i|PLA]$ vs. $[LLA|i]$. This family of functions also enjoys the following very useful property (proved in the Appendix).

Property 2. *For any $0 \leq i < j < N$, $f_i(LLA) \neq f_j(LLA)$ for every LLA.*

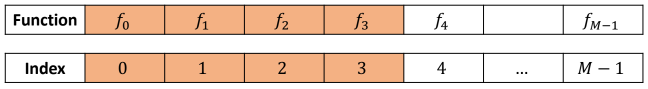
The importance of Property 2 is that a single LLA does not return to the same PLA before reaching index $i = N$. This allows the wear-leveling scheme using this mapping to utilize the endurance of all N PLAs even if a single LLA is written by the host.

2.2 Sliding Window of Mapping Indices

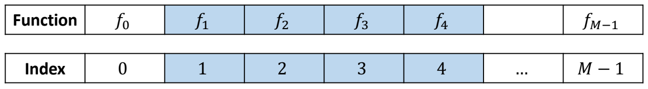
The large number ($\geq N$) of mapping indices supported by the proposed mapping functions is clearly useful for effective spreading of the write load throughout the entire device. However, toward limiting the resources consumed by the mapping, we restrict all LLAs to have mapping indices in a subset of S consecutive indices. This subset changes as a sliding window throughout the device lifetime. That is, the mapping-index set is $\{\text{base}, \text{base}+1, \ldots, \text{base}+S-1\}$, for some integer base, and every LLA has a mapping index

$$i = \text{base} + \text{offset}_i, \tag{2}$$

where $\text{offset}_i \in \{0, \ldots, S-1\}$. This allows the mapping architecture to keep the value of base in a global register, and represent the index i compactly as offset_i, which takes only $\log_2 S$ bits. Figure 2a depicts this restriction of indices to a sliding window of size $S = 4$. S is a design parameter of the architecture: large S allows more flexibility for selective remapping, but also increases the complexity and/or costs of maintaining the mapping. We defer to Section 4 the detailed discussion of the effect of S on mapping complexity. In the mean



(a) Mapping sliding window.



(b) Mapping sliding window after a movement.

Figure 2: Sliding window of active mapping indices, for the case $S = 4$. (a) Initial set in shaded orange when base $= 0$. (b) After incrementing to base $= 1$, the set moves to the window in shaded blue.

time, we note a practical disadvantage of using offset$_i$ (as in (2)) for representing the index, due to the need to update offset$_i$ when the window slides to a subsequent base, even if i is unchanged. Instead, we propose an alternative compact representation: $\bar{i} = i \bmod S$, which can be used to recover i using the formula

$$i = \text{base} + ((\bar{i} - \text{base}) \bmod S). \tag{3}$$

It is clear that $\bar{i}$ remains unchanged if i is unchanged, even if base is increased. For example, an LLA mapped with $i = 3$ in Figure 2a has $\bar{i} = 3$, which remains the same even after the window movement to base $= 1$ in Figure 2b. In this latter state, i can be recovered by (3): $i = 1 + (3 - 1) \bmod 4 = 3$.

2.3 Selective Remapping

The unique feature of the proposed architecture is that different LLAs can be mapped by different mapping functions (mapping indices). This feature allows *selective remapping* of heavily written LLAs by incrementing their mapping index, while keeping other LLAs at their current mapping indices and physical locations. Thanks to the device over-provisioning ($N > K$), it is possible to remap an LLA with minimal change to the mapping of other LLAs. A heavier remapping operation, called *catch-up*, occurs when the remapped LLA's mapping index is incremented beyond the current index window. In this case, the index window needs to shift, and with it will move all the LLAs that are currently mapped by indices below its new base index. However, since $S \gg 1$, catch-up events reflect a minuscule minority of the remapping events. We give some more details on the remapping operations, starting with how remapping events are triggered.

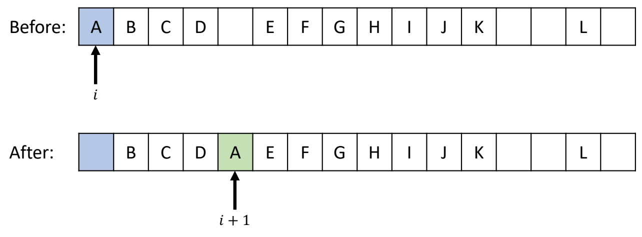
2.3.1 Remapping trigger. Selective remapping warrants the definition of a trigger event for moving *a specific LLA* from its current PLA. A global write counter, used in most prior wear-leveling architectures (e.g., [14]), would not suffice in this case. Informally, we remap an LLA written by the host if its current PLA has reached a wear level that holds the risk of its premature failing. Toward this end, we specify a wear threshold $\phi < w_{max}$, with the following policy: a *host write* to an LLA mapped to a PLA that had *exceeded ϕ writes* will be written after remapping the LLA to a different PLA. Note that the policy is only applied to host writes; a write that is part of a remapping operation will *not* trigger an additional remapping, even if the written PLA exceeded the threshold. This differentiation makes the remapping procedures (discussed next) simpler and more deterministic. The value of ϕ is an optimization variable, set to vacate a worn PLA "just in time" to keep it usable for all future remappings. In Section 3 we specify a formula for the value of ϕ, derived based on analysis of ECC-Map that is included in the appendix. Although ϕ is given as a count of physical writes, implementing the remapping trigger does not require maintaining PLA write counters (which would be expensive). Instead, reaching a threshold of ϕ can be detected by a reliability measurement of the PLA, for example by counting the number of bit errors corrected by the decoder of the data error-correcting codes.

2.3.2 Regular remapping. The vast majority of remapping operations follow the simple procedure we describe next. When LLA triggers remapping, it is moved from mapping index i to $i + 1$. If $f_{i+1}(LLA)$ is an unused PLA, the remapping is complete – we call this a *non-colliding* regular remapping. In a *colliding* regular remapping, before writing LLA to $f_{i+1}(LLA)$, LLA' currently mapped to this PLA with index j is remapped to index $j + \delta$, and δ is the smallest positive integer such that $f_{j+\delta}(LLA')$ is an unused PLA. The procedure guarantees the movement of LLA to index $i + 1$, and in case of collision, moves LLA' out to a free PLA. The rationale behind giving LLA the priority over LLA' is to minimize the index increase of *host-written* LLAs, thus allowing more writes before an LLA exits the index window.

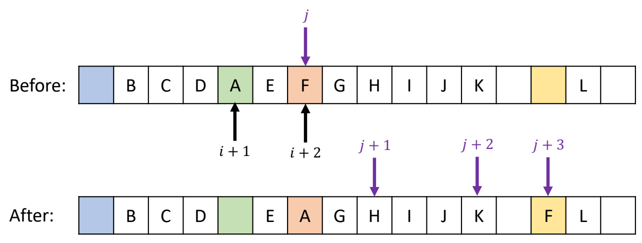
2.3.3 Catch-up remapping. When an LLA needs to move during regular remapping to an index equal to or greater than base $+ S$, a catch-up procedure is invoked. In the catch-up procedure we first set a new base value greater than the current one, and then remap every LLA with index smaller than the new base to an index greater or equal to it. The amount by which base is shifted, as well as the new indices chosen for the catching-up LLAs, are a matter for optimization. In this work we simply set base $\longleftarrow$ base $+ S$, and remap all LLAs with smaller indices to the new base index. Other catch-up algorithms (not used in this work) may alternatively add less than S to base, and/or move LLAs to indices strictly beyond the new base.

Figure 3a and Figure 3b illustrate the operations of regular remapping described above. The plotted arrays represent the PLA space of the device, and the capital letters in the array are the LLAs mapped to the corresponding PLAs. Initially, LLA A is mapped by index i as shown at the top part of Figure 3a. Upon its triggered remapping, A moves to its PLA position at the bottom part by incrementing its index to $i + 1$. In this remapping there is no collision with another LLA. Figure 3b shows the next remapping of A, in which there is collision with LLA F; to free the PLA to A, F is moved to the PLA mapped to it by index $j + 3$, because the lower indices $j + 1$ and $j + 2$ map to used PLAs.

Figure 4 illustrates the catch-up procedure described above. For $S = 4$, the figure displays the mapping index of each LLA. Initially, base $= 0$ and A has index 3 (top part). Upon remapping of A (bottom part), its index is incremented to 4, which falls outside the current window $\{\text{base} = 0, 1, 2, 3 = S - 1\}$, thus invoking catch-up. This example implements the simple algorithm we use in this work: setting base $\longleftarrow$ base $+ S = 4$, and updating all LLAs to the new base.



(a) Regular remapping - non-colliding case.



(b) Regular remapping - colliding case.

Figure 3: Illustration of regular remapping. (a) The non-colliding case. In this case, A moves from index i to $i+1$ and reaches a free PLA. (b) The colliding case. In this case, the next index $i+2$ maps A to a used PLA, thus requiring the movement of F to a subsequent index that maps it to a free PLA.

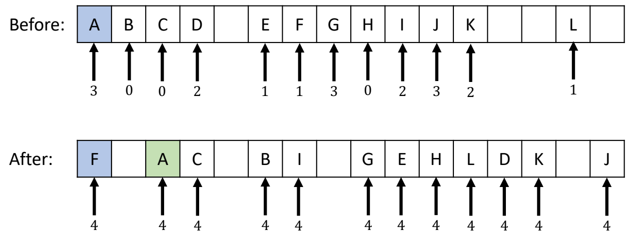


Figure 4: Illustration of the catch-up procedure. Initially, base $=0$ **and all LLAs have indices in the range** $\{0,1,2,3\}$**. Then A is remapped to index** 4**, invoking a catch-up procedure leading to** base $=4$ **and all LLAs mapped with index** 4**.**

The final ingredient of the proposed mapping architecture is index randomization, applied for the purpose of hiding the instantaneous mapping functions from an adversary generating the write workload.

2.4 Mapping-index Randomization

As we make the standard assumption that the mapping functions used by the architecture are publicly known, an adversary may be able to track the mapping of LLAs and issue writes to those mapped to high-wear PLAs. To prevent this, we add a pseudo-random transformation between the running mapping indices $1,2,3,\dots,N-1$ and the actual mapping numbers fed to the mapping functions (forward and inverse) of Figure 1a and Figure 1b.

For the transformation we use a standard linear-feedback shift register (LFSR) [2, 18], which is initialized to a random seed generated internally by the device. The random seed is the output mapping number corresponding to the initial mapping index 1 (we skip index 0 in the randomized setting). Then, each update of the register using the linear feedback gives the output mapping number of the subsequent mapping index. To maintain Property 2 for the output mapping numbers, we use an m-bit LFSR with period $2^m-1=N-1$. This guarantees that no two indices in $1,\dots,N-1$ have the same output mapping number, and fixes the $k-2m$ most significant bits of the output mapping number to the same value for all indices $1,\dots,N-1$, thus implying $f_{\text{LFSR}(i)}(LLA)\neq f_{\text{LFSR}(j)}(LLA)$ similarly to Property 2.

3 EVALUATION AND RESULTS

Before evaluating the proposed mapping architecture, we specify the formula we use to set the threshold parameter ϕ (See Section 2.3.1), as a function of the architecture parameters N, w_{max}, S. The formula is based on a theoretical analysis of ECC-Map for the 1-LLA workload that repeatedly writes to a single LLA until reaching the device end of life. The detailed analysis can be found in the appendix. We denote $\alpha\triangleq\phi/w_{max}$ as the fractional threshold, and set α to be the following

$$\alpha_{\text{opt}}=\begin{cases}1-\frac{N}{Sw_{max}}, & \frac{N}{w_{max}}<\frac{S}{3}\\ \frac{2}{3}, & \text{otherwise}\end{cases}\tag{4}$$

The subscript "opt" is used to mark the fact that this α maximizes the utilization on the 1-LLA workload, according to the model and its analysis in the appendix.

3.1 Evaluation

Implementation. To evaluate the performance of the proposed architecture, we implemented all of its ingredients in a Python-based discrete-event simulator. The simulator accepts an arbitrary write workload, and runs it through the proposed device mapping layer, including exact management of the indexed mapping functions, and performing all remappings (regular and catch-up). Upon reaching the device end of life, that is, when a PLA first exceeds w_{max} writes, the simulator stops and records the utilization value for this workload. Using a software simulator allows us to examine the device performance in a large variety and broad range of system variables. The principal system variable in our evaluation, which turns out to be the key performance determinant, is the *size-to-endurance ratio* N/w_{max}. In general, as N/w_{max} grows, the more "difficult" it becomes for a given mapping architecture to level the wear. This fact is observed in [14], with the inequality $1/\psi>N/w_{max}$, where the left-hand side is the frequency of internal-copy writes required by the SG architecture. The same ratio N/w_{max} also appears (twice) in (4). The principal dependence on the ratio N/w_{max} allows us to use relatively small values in most tests ($N=1024$, varying w_{max}), significantly speeding up the evaluation. To prove that the absolute values of N,w_{max} are secondary to their ratio, our evaluations include tests we repeat with 4 and 16 times larger N,w_{max}, showing no significant difference. We thus expect that much larger commercial devices with these size-to-endurance ratios – for example, a $N/w_{max}=2$ device with 2G-lines and endurance $1e9$ – will perform similarly.

In addition to N/w_{max}, other system variables we examine in our evaluation are the window size S, the spare factor ρ, and the trigger threshold ϕ. For convenience, we list in Table 1 the default values we use for these system variables, unless noted otherwise (each result typically varies one variable, leaving the rest to their default values).

system variable	default value
Size-to-endurance ratio N/w_{max}	0.5
Window size S	32
Spare factor ρ	20%
Trigger threshold ϕ	set by (4) to $\alpha_{\text{opt}} w_{max}$

Table 1: Default values of system variables.

Comparison. In addition to studying the performance of the proposed architecture, this section compares this performance to the three state-of-the-art wear-leveling architectures for PCM: the SG and RBSG architecture [14], where the latter adds region partition on the former, and the region-based secure-PCM main-memory architecture [15] (Sec-Mem). The last of the three works by dynamically remapping a full region every certain number of host writes to it since the last remapping. Note that dynamic region mapping requires a mapping table with size linear in the number of regions. While there are follow-up works enhancing these architectures in different ways, these three works are the best known for the standard device model we consider here. Therefore, we expect to see similar advantage over other variants, which also use global or region write counters as a trigger for remapping. For the SG architecture we use a device with the same logical capacity K, and a single spare PLA, as specified in [14]; for RBSG we use the same K and N parameters as ECC-Map. The secure PCM architecture does not need spare, thus it is used with $N = K$ (and the same K). Note that the comparison is fair even if the values of N are not equal, because the utilization metric penalizes the increased N appropriately.

Measurements. We ran different write workloads, and for each architecture we counted the total number of writes (host and physical) the device served until its end of life. We recorded several performance metrics: the *(logical) utilization* (1), the *number of host writes*, and the *number of physical writes*. We repeated each test five times and averaged the results to smooth out the workload randomness.

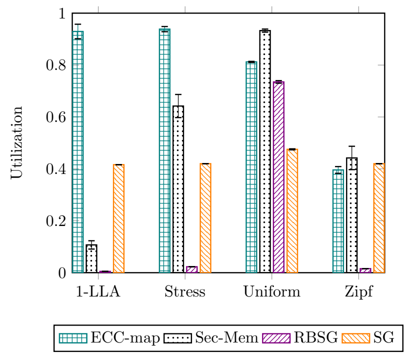
Workloads. We tested four write workloads in our evaluations: 1) the *1-LLA* workload, 2) the *stress* workload, 3) the *uniform* workload, and 4) the Zipfian distribution *Zipf*. In 1), we randomly choose a single LLA, and write to it repeatedly until reaching end of life. In 2), we randomly pick a 3% fraction of the LLAs and write only to them, where the selection within the set is uniform. In 3), each write draws an LLA uniformly from the entire space. In 4), each write draws an LLA from the whole address space with a non-uniform selection that follows the distribution $p(i, K) = \frac{1/i}{\sum_{n=1}^{K} 1/n}$, where i is the LLA's sequence number and K is the number of LLAs. The 1-LLA workload is the key motivation of this work, hence it will be the focus of the evaluation. The stress and Zipf workloads model other challenging write patterns that the device needs to handle, and the "easier" uniform workload is included mainly as reference, since it is handled well by prior wear-leveling architectures.

3.2 Results

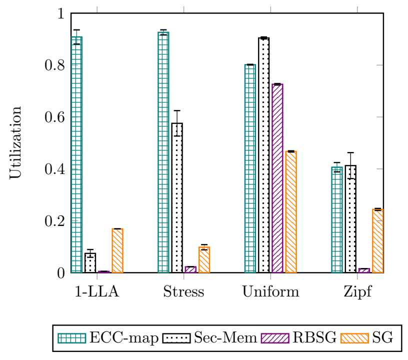
We first use the default values from Table 1 and plot in Figure 5a the utilizations of the four architectures for each of the four workloads. It is first observed that ECC-Map significantly outperforms the three prior architectures on the 1-LLA workload. RBSG's performance is satisfactory only on the uniform workload, and SG's is lower than ECC-Map on all workloads except Zipf, on which it is very close to ECC-Map. Sec-Mem's performance on the stress workload is about a 1/3 worse than ECC-Map. On the uniform workload all architectures have good performance, as expected, with Sec-Mem slightly ahead of ECC-Map coming second. We continue the experiment with progressively larger size-to-endurance ratios N/w_{max}. Recall from the discussion in Section 3.1 that larger ratios are in general more difficult to wear-level. Moving from the default value of 0.5 in Figure 5a, we plot in Figures 5b-5e the utilizations for four larger values of N/w_{max}, each time multiplying it by 2. We get these ratios by fixing N and halving w_{max} successively. The value of ϕ is calculated using (4) for each tested ratio. We indeed see that increasing the size-to-endurance ratio decreases the utilizations on the 1-LLA and stress workloads. However, this decrease is much more graceful in ECC-Map than in Sec-Mem, while both SG and RBSG have near-zero utilizations on these workloads starting from $N/w_{max} = 2$. The performance of ECC-Map on the Zipf workload even improves with larger size-to-endurance ratios, for reasons that will be explained later in Section 3.2.3. The very low utilization values of both SG/RBSG and Sec-Mem throughout Figure 5 mean that these architectures cannot be used by devices with such size-to-endurance ratios.

3.2.1 Dependence on the absolute device size. In the next experiment, we examine the (in)sensitivity of the results to the absolute values of N,w_{max}, thus corroborating our claim that performance is determined by their ratio N/w_{max}. Toward that, we ran the same workloads with the same ratios and three different device sizes $N = 1024, 4096, 16384$ (with corresponding w_{max} values). The results are recorded in Table 2, showing almost identical values of (logical) utilization between the different sizes.

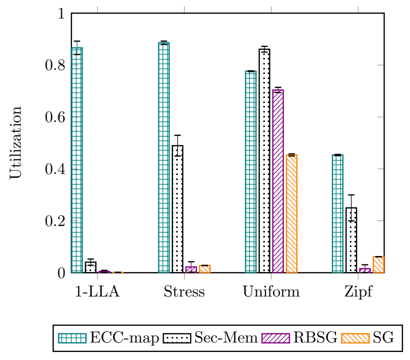
3.2.2 Dependence on the window size S. In Figure 6, we plot the utilization's dependence on the window-size parameter S. Recall that S controls the mapping richness/complexity, so it is important to examine its effect on performance. We ran the workloads using the default device parameters, each time implementing a different window size $S = 16, 32, 64, 128$. First, the results show that for all values of S and all workloads, the architecture achieves significant utilization (for comparison, we plot the RBSG 1-LLA utilization as a horizontal line). It can be observed that significant improvement is offered to the 1-LLA workload when increasing S from 16 to 32, while subsequent increases give more modest advantages. That means that for these device parameters, $S = 32$ may be the right compromise between performance and mapping cost. It can also be seen that the uniform and stress workloads are less sensitive to the value of S. This is because more balanced workloads have more balanced mapping-index distributions, and thus fewer catch-up remappings even when S is small. The stress workload sees some small utilization decrease in $S = 128$, which can be attributed to the fact that ϕ is optimized for the 1-LLA workload. In a real



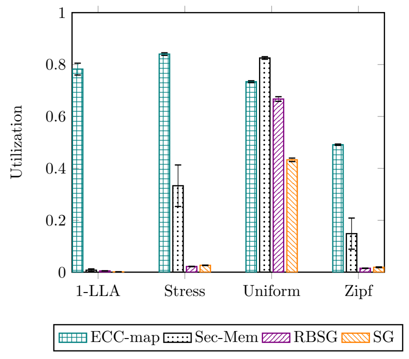
(a) $\frac{N}{w_{max}} = 0.5$



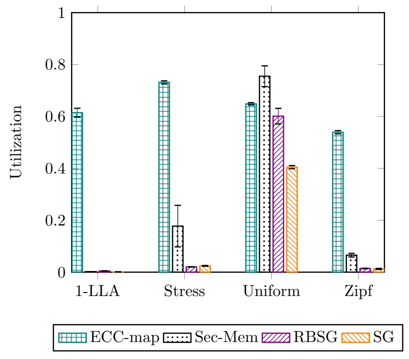
(b) $\frac{N}{w_{max}} = 1$



(c) $\frac{N}{w_{max}} = 2$



(d) $\frac{N}{w_{max}} = 4$



(e) $\frac{N}{w_{max}} = 8$

Figure 5: Utilization as a function of the size-to-endurance ratio.

Workload	Device size - N	w_{max}	ϕ	Logical utilization (1)	Host writes	Physical writes
1-LLA	1024	128	96	0.61	80540	108779.8
	4096	512	384	0.61	1281893.6	1777136
	16384	2048	1536	0.61	20443371	28573809.8
Uniform	1024	128	96	0.65	85005.2	100765.4
	4096	512	384	0.65	1368310.4	1702714.2
	16384	2048	1536	0.65	21910283.2	28316701.2
Stress	1024	128	96	0.73	95844.6	115641.8
	4096	512	384	0.74	1559924.4	1952942.6
	16384	2048	1536	0.75	24888046.8	31827576
Zipf	1024	128	96	0.55	71901.2	86187.2
	4096	512	384	0.56	1184265.8	1495514.2
	16384	2048	1536	0.54	18242031.2	23847574.6

Table 2: Comparing detailed run statistics (averaged) for three different device sizes with the same ratio $N/w_{max} = 8$.

implementation of the architecture, one may choose to set ϕ to jointly optimize for different workloads, but good utilization is achieved even with the simple formula used in this work.

3.2.3 Dependence on the remapping threshold ϕ. To expand on the issue of optimizing the threshold ϕ, we point back to Figure 5 and note that for $N/w_{max} = 0.5$, the value of the fractional ratio ϕ/w_{max} according to (4) is as high as $1-1/64$. Such high thresholds, while optimal for the 1-LLA workload, limit the performance on the Zipf workload. Thus, a possible solution is to set ϕ/w_{max} as the minimum between the outcome of (4) and a predefined limit, e.g., 0.8. The effect of this is demonstrated in Figure 7, comparing the performance of ECC-Map with and without this modification. It can be seen that the Zipf utilization increased from around 0.4 (as also seen in Figure 5a) to over 0.7. At the same time, the modification did decrease the 1-LLA (and stress) utilizations, but not significantly so.

To validate the correctness of the ϕ_{opt} derived in (4), we next want to see the utilization as a function of ϕ. We define $\phi_{\text{opt}} \triangleq \alpha_{\text{opt}} \cdot w_{max}$, and for convenience plot in Figure 8 the utilization as a function of $\phi/\phi_{\text{opt}} - 1$. The x-axis point 0 represents the value of $\phi = \phi_{\text{opt}}$

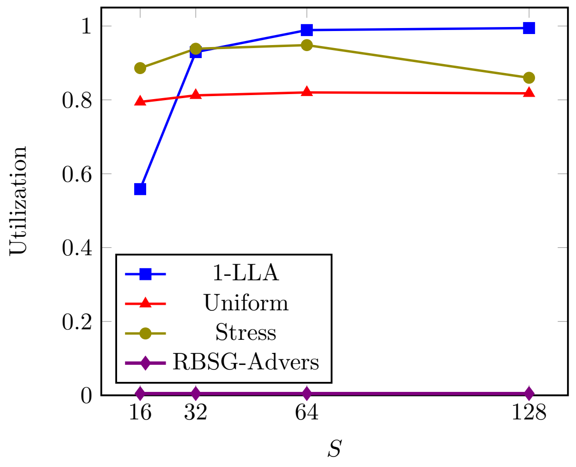


Figure 6: Utilization as a function of the mapping window size S.

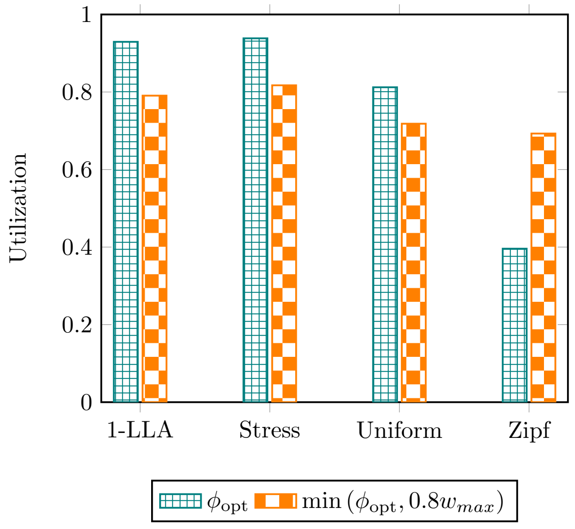


Figure 7: ECC-map with $\phi = \phi_{\text{opt}}$ vs. with $\phi = \min(\phi_{\text{opt}}, 0.8w_{max})$.

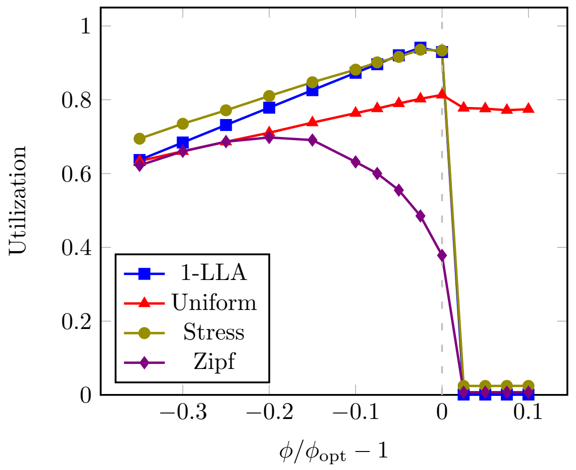


Figure 8: Utilization as the threshold-trigger ϕ is varied from its optimized value.

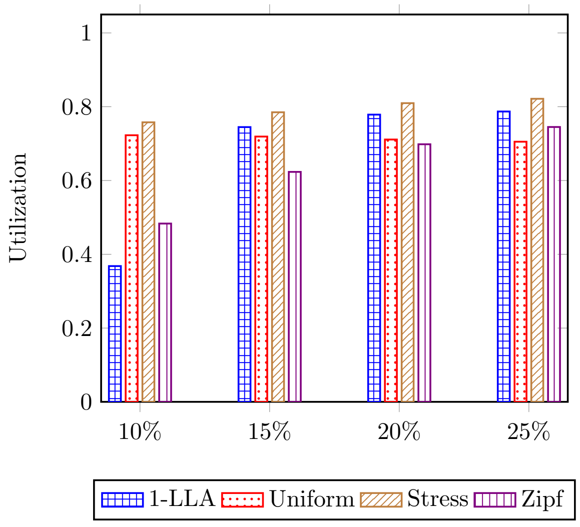


Figure 9: Utilization for different values of spare factor.

as specified by (4), where the values to the left and right of that point represent smaller and bigger ϕ values, respectively. One can see that utilization on the 1-LLA (and stress) workload is maximized close to the value of 0, indicating the correctness of the analysis leading to the specified value ϕ_{opt}. Further, it is seen that a small increase in ϕ may harm utilization significantly (for all workloads except the uniform), while a decrease is largely harmless and even helpful for the Zipf workload. For good performance in all workloads, this plot motivates adopting the point of $\phi/\phi_{\text{opt}} = 0.8$, which we also use in the next sub-section. The plot also gives an important insight for design purposes: since one may have only an estimate of the line wear, it is important that it will be an *over*estimate, such that harmless premature remappings are favored over late ones that decrease utilization.

3.2.4 Dependence on the spare factor. Finally, in Figure 9 we plot the utilization for four different values of spare factor $\rho = 0.1, 0.15, 0.2, 0.25$. N is fixed to its default value, and K is varied to get the corresponding spare factors. Recall from Section 2.3.2 that using spare PLAs helps having non-colliding regular remappings, which reduces the amount of internal-copy writes and slows down the advancement of mapping indices. Firstly, the plot shows that even with spare factor as small as 0.1 the utilization is significant. That said, increasing it to 0.15 gives substantial advantage in the more challenging workloads of 1-LLA and Zipf. Increasing it further exhibits diminishing returns. We reiterate the fact that the utilization metric takes into account the added cost of the spare PLAs, thus giving a fair comparison across different spare factors. The fact that the utilization is normalized by N means that it is not necessarily increasing with the spare factor, as seen in the decreasing trend for the uniform workload. This decrease can be attributed to the growing number of remappings needed to claim unused endurance in more spare PLAs.

4 IMPLEMENTATION CONSIDERATIONS

Toward using the ECC-Map architecture in a real memory device, this section expands upon important details needed for efficient implementation. Primarily, it discusses ways by which the window-size parameter S translates to bounded implementation complexity.

4.1 Achieving Low-Complexity Mapping

The basic requirement from the architecture to support read and write operations is to be able to map any LLA to its current PLA. In ECC-Map, this requirement is reduced to *mapping an LLA to its current mapping index*, thence finding the PLA by simple invocation of the mapping function. Using a global register for the current base mapping index, this requirement is further reduced to mapping an LLA to its offset from the base index – see (3) in Section 2. This last reduction is the key promise toward achieving efficient mapping: representing the offset index only requires $\log_2 S$ bits (rounded upward to the next integer), while representing the PLA or the full mapping index requires $\log_2 N$ bits. Recall that N is a device parameter that grows with the scaling of the memory size, while S is a small fixed constant. For example, in a 500GB memory device with line size of 512B, we have $N \approx 2^{30}$, while the "recommended" window-size parameter in Section 3 is $S = 32 = 2^5$.

In addition to the forward (LLA→PLA) mapping, the remapping operations described in Section 2.3 require inverse-mapping a PLA to its mapping index. The simplest and lowest-cost way to support inverse mapping is by storing the offset index on the line itself on the media, alongside the data. The cost is negligible when the line size is much larger than $\log_2 S$. Next, we describe three possible implementation approaches of the forward mapping in ECC-Map.

4.1.1 *Reduced-size mapping table.*

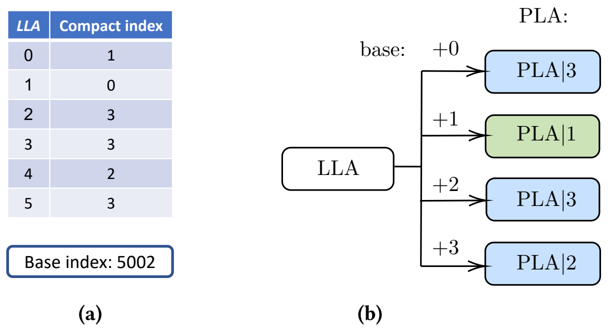
The straightforward way to implement forward mapping in the proposed architecture is through a table mapping each LLA to its offset mapping index, see Figure 10a. This already gives a major advantage over full-indirection mapping: for example, in a 500GB device with line size 512B and $S = 32$ this saves $1 - \log_2 S / \log_2 N = 83.25\%$ of the table memory cost, where $N = 500e9/512$.

4.1.2 *Advanced mapping data structures.*

The fact that S is a relatively small constant opens the way to devising clever data structures for mapping LLAs to offsets, which will be more economical than a table. Such data structures are beyond the scope of this work, but we give a simple example case to clarify this direction. Suppose that the base mapping index is defined to be the default, and the mapping data structure only needs to record LLAs having other indices. Then we need a data structure that records membership in the remaining $S - 1$ offsets, and whose size is proportional to the number of LLAs not using the default index. Keeping this data structure size under the constraints of the pre-allocated memory is an interesting optimization problem. In unbalanced workloads (such as 1-LLA and stress), we naturally get that the majority of LLAs use the same mapping index, and in more balanced workloads we can afford triggering special catch-up remappings for consolidating the mapping data structure.

4.1.3 *Mapping with no data structure.*

It is possible to implement the forward mapping without using any data structure, except for the global base register. Recall that for the inverse mapping we



LLA	Compact index
0	1
1	0
2	3
3	3
4	2
5	3

Base index: 5002

Figure 10: (a) The straightforward forward-mapping implementation: Reduced-size mapping table, example for $S = 4$. A global base **register, and $\log_2 S$ offset bits for each LLA. (b) Most economical forward-mapping implementation: Mapping with no data structure, example for $S = 4$. First calculating the PLAs $\{f_{\text{base}+i}(LLA)\}_{i=0}^{3}$, then reading from the media the PLAs' offset values, and choosing the one that matches the offset used in the forward mapping (the PLA marked in green).**

store in every PLA its offset index. Thus, given an LLA, we can find all S PLAs that may be mapped to it for the current base, and read from each one the $\log_2 S$ inverse-mapping bits. The S PLAs are $\{f_{\text{base}}(LLA), f_{\text{base}+1}(LLA), \ldots, f_{\text{base}+S-1}(LLA)\}$, and it can be verified[1] that there is exactly one $PLA = f_i(LLA)$ whose stored inverse-mapping offset bits represent the index i according to (3). This matching PLA is found as the current mapping of the LLA. See Figure 10b. While this method requires accessing multiple physical locations on the memory media, only a small number $S \log_2 S$ of bits are read in total, and this operation may also be parallelized depending on the memory technology.

4.2 Efficient LFSR Transformation

Recall from Section 2.4 that the running index i calculated in (3) undergoes an LFSR transformation before entering the index fields of the mapping functions in Figure 1a and Figure 1b. To perform these transformations, the device has to have efficient access to the values $LFSR(i)$, for $i = \text{base}, \ldots, \text{base} + S - 1$. This can be achieved either by maintaining a cache holding these S values, or by efficient cycling of the LFSR back and forth in this range.

5 RELATED WORK

Wear leveling is a key problem in designing and deploying wear-limited memories, and has thus attracted considerable prior research attention. The problem initially arose in Flash-based memories, and later considered for phase-change memories (PCM) and related emerging technologies for persistent memories. In Flash memory the problem is somewhat simplified, at least from the mapping perspective, thanks to the common employment of full indirection in the flash translation layer (FTL). In persistent memories based on PCM (and related) technologies, full indirection is not a likely option, due to the lack of need for out-of-place writing, and the smaller line sizes.

[1]otherwise violating the proven property that an LLA is mapped to exactly one PLA.

Prior wear-leveling solutions, for both Flash and PCM devices, use a variety of techniques, at different parts of the memory stack: from the operating system to the physical representation of cell levels. We now briefly mention a non-exhaustive sample of these techniques.

5.1 Wear-leveling and Related Techniques for PCM Devices

The most celebrated PCM wear-leveling architecture is Start-Gap [14], thanks to its simplicity and extremely efficient mapping layer. The key technique used in Start-Gap [14] is periodical *line shifting*, called therein *gap movements*. In addition, it proposes to divide the device to *regions* to better mitigate extremely unbalanced workloads (though at the cost of unused endurance in some regions). In [26] and [15], line shifting is complemented by *region swapping* for improved wear spreading. While region swapping helps, its flexibility for claiming unused endurance depends on the region size, and fine region partition requires large mapping tables. Region swapping is further enhanced in [27] by considering endurance variation among different regions for selecting the swap target. Exploiting variation is a useful technique, complementary to the design of the mapping architecture, and can also enhance the proposed ECC-Map architecture. Another technique used in almost all wear-leveling architectures is *address randomization* for hiding the mapping from an adversary, as we implement here in ECC-Map.

Additional works address wear leveling as part of larger architectural settings, building on the techniques mentioned in the previous paragraph. [3–5] incorporate wear leveling into the operating-system stack; [24, 25] combine PCM and DRAM (the latter having much higher endurance); and [23] proposes a novel hardware address decoder (PRAD) that can help in wear leveling (among other things).

A vastly studied approach, related to wear leveling, is *wear reduction*. [6] proposes a physical-writing mechanism for PCM that reduces the write wear. [10] uses information from the L1 cache to write only modified data to the PCM media. A similar objective is pursued in [8], which in addition presents a wear-leveling scheme in PCM when it acts as a cache. Wear reduction techniques are extremely useful in practice, and can similarly enhance the performance of ECC-Map.

5.2 Wear-leveling Techniques for Flash Devices

Flash-based memories differ from PCM and newer persistent memories in their internal structure of large update units (called blocks), each comprising many lines (known as pages). Due to this structure, Flash wear leveling is done at the larger block granularity, and assuming the availability of a translation layer supporting flexible logical to physical mapping. Most of the techniques use a table that tracks the wear of each data block, hence picking low-wear blocks for the incoming host writes. [16] considers the endurance variability among different blocks, and tabulates block reliability statistics based on measuring program error rate. ECC-Map can also be extended to consider variability, by setting variable ϕ thresholds for different parts of the device. [19] further extends the reliability estimation by considering retention errors through time measurements between consecutive program cycles. [22] suggests using multiple block remapping thresholds, reducing the number of writes between remappings as the device ages.

6 CONCLUSION

In this work, we present ECC-Map, a novel wear-leveling scheme for persistent memories that can handle even the most unbalanced workloads. A family of efficient functions based on ECC encoders provides flexible and economical mapping, and enables remapping operations that are more targeted to the incident workload. ECC-Map's remapping algorithms are extremely simple, which is important for implementation on device controllers. Toward that, many interesting topics are left for future work. Among them: 1) the organization of the mapping meta-data on the memory media, 2) the optimization and scheduling of remapping operations, and 3) further improvements to the proposed mapping algorithms, offering interesting tradeoffs among different workloads.

7 ACKNOWLEDGEMENT

This work was supported in part by the Israel Science Foundation under grant number 2525/19.

REFERENCES

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

[2] Paul H. Bardell, William H. McAnney, and Jacob Savir. *Built-in Test for VLSI: Pseudorandom Techniques.* Wiley-Interscience, 1987.

[3] Yu-Ming Chang, Pi-Cheng Hsiu, Yuan-Hao Chang, Chi-Hao Chen, Tei-Wei Kuo, and Cheng-Yuan Michael Wang. Improving PCM Endurance with a Constant-Cost Wear Leveling Design. *ACM Trans. Des. Autom. Electron. Syst.*, 22(1), jun 2016.

[4] Chi-Hao Chen, Pi-Cheng Hsiu, Tei-Wei Kuo, Chia-Lin Yang, and Cheng-Yuan Michael Wang. Age-Based PCM Wear Leveling with Nearly Zero Search Cost. In *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, page 453–458, New York, NY, USA, 2012. Association for Computing Machinery.

[5] Sheng-Wei Cheng, Yuan-Hao Chang, Tseng-Yi Chen, Yu-Fen Chang, Hsin-Wen Wei, and Wei-Kuan Shih. Efficient Warranty-Aware Wear Leveling for Embedded Systems With PCM Main Memory. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(7):2535–2547, 2016.

[6] Sangyeun Cho and Hyunjin Lee. Flip-N-Write: A Simple Deterministic Technique to Improve PRAM Write Performance, Energy and Endurance. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, page 347–357, New York, NY, USA, 2009. Association for Computing Machinery.

[7] George C. Clark and J. Bibb. Cain. *Error-correction coding for digital communications.* Plenum Press New York, 1981.

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

[9] Miguel Angel Lastras-Montano and Kwang-Ting Cheng. Resistive random-access memory based on ratioed memristors. *Nature Electronics*, 1(8):466–472, 2018.

[10] Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger. Architecting Phase Change Memory as a Scalable Dram Alternative. *SIGARCH Comput. Archit. News*, 37(3):2–13, jun 2009.

[11] Dongzhe Ma, Jianhua Feng, and Guoliang Li. A survey of address translation technologies for flash memories. *ACM Comput. Surv.*, 46(3), jan 2014.

[12] Frederic P. Miller, Agnes F. Vandome, and John McBrewster. *Cyclic Redundancy Check: Computation of CRC, Mathematics of CRC, Error Detection and Correction, Cyclic Code, List of Hash Functions, Parity Bit, Information ... Cksum, Adler- 32, Fletcher's Checksum.* Alpha Press, 2009.

[13] Ardavan Pedram, Stephen Richardson, Mark Horowitz, Sameh Galal, and Shahar Kvatinsky. Dark Memory and Accelerator-Rich System Optimization in the Dark Silicon Era. *IEEE Design & Test*, 34(2):39–50, 2017.

[14] Moinuddin K. Qureshi, John Karidis, Michele Franceschini, Vijayalakshmi Srinivasan, Luis Lastras, and Bulent Abali. Enhancing lifetime and security of

PCM-based Main Memory with Start-Gap Wear Leveling. In *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 14–23, 2009.

[15] Andre Seznec. A Phase Change Memory as a Secure Main Memory. *IEEE Computer Architecture Letters*, 9(1):5–8, 2010.

[16] Xin Shi, Fei Wu, Shunzhuo Wang, Changsheng Xie, and Zhonghai Lu. Program error rate-based wear leveling for NAND flash memory. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1241–1246. IEEE, 2018.

[17] Tae-Sun Chung and Dong-Joo Park and Sangwon Park and Dong-Ho Lee and Sang-Won Lee and Ha-Joo Song. A survey of flash translation layer. *Journal of Systems Architecture*, 55(5):332–343, 2009.

[18] Thomas E. Tkacik. A hardware random number generator. In Burton S. Kaliski, çetin K. Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 450–453, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[19] Debao Wei, Liyan Qiao, Xiaoyu Chen, Mengqi Hao, and Xiyuan Peng. SREA: A self-recovery effect aware wear-leveling strategy for the reliability extension of NAND flash memory. *Microelectronics Reliability*, 100-101:113433, 2019. 30th European Symposium on Reliability of Electron Devices, Failure Physics and Analysis.

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

[21] Ming-Chang Yang, Yu-Ming Chang, Che-Wei Tsao, Po-Chun Huang, Yuan-Hao Chang, and Tei-Wei Kuo. Garbage collection and wear leveling for flash memory: Past and future. In *2014 International Conference on Smart Computing*, pages 66–73, 2014.

[22] Yuan Hua Yang, Xian Bin Xu, Shui Bing He, Fang Zhen, and Yu Ping Zhang. WLVT: A Static Wear-Leveling Algorithm with Variable Threshold. In *Advanced Materials Research*, volume 756, pages 3131–3135. Trans Tech Publ, 2013.

[23] Leonid Yavits, Lois Orosa, Suyash Mahar, João Dinis Ferreira, Mattan Erez, Ran Ginosar, and Onur Mutlu. WoLFRaM: Enhancing Wear-Leveling and Fault Tolerance in Resistive Memories using Programmable Address Decoders. In *2020 IEEE 38th International Conference on Computer Design (ICCD)*, pages 187–196, 2020.

[24] HanBin Yoon, Justin Meza, Rachata Ausavarungnirun, Rachael A. Harding, and Onur Mutlu. Row buffer locality aware caching policies for hybrid memories. In *2012 IEEE 30th International Conference on Computer Design (ICCD)*, pages 337–344, 2012.

[25] Wangyuan Zhang and Tao Li. Characterizing and mitigating the impact of process variations on phase change based memory systems. In *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 2–13, 2009.

[26] Ping Zhou, Bo Zhao, Jun Yang, and Youtao Zhang. A Durable and Energy Efficient Main Memory Using Phase Change Memory Technology. *SIGARCH Comput. Archit. News*, 37(3):14–23, jun 2009.

[27] Wen Zhou, Dan Feng, Yu Hua, Jingning Liu, Fangting Huang, and Pengfei Zuo. Increasing lifetime and security of phase-change memory with endurance variation. In *2016 IEEE 22nd International conference on parallel and distributed systems (ICPADS)*, pages 861–868. IEEE, 2016.

APPENDIX / SUPPLEMENTARY MATERIAL

Discussion of the Overall Device Operation

As detailed in Section 2.3, host writes go directly to their mapped PLAs so long that the PLA's write count is below ϕ, and trigger remapping otherwise. That means that a PLA can reach its end-of-life of w_{max} writes only during a remapping operation. There are two factors by which the utilization (1) is degraded from its theoretical limit of 1: 1) remaining unused writes of PLAs, and 2) internal-copy writes during remapping. There is at most one internal-copy write in a regular remapping operation: one when the remapping is colliding, and none when it is non-colliding. Considering $f_i(\cdot)$ as a random function whose output is uniformly drawn from $\{0,\ldots,N-1\}$, the probability that a regular remapping is colliding equals $K/N = 1-\rho$. Hence the number of internal-copy writes can be reduced by increasing the spare factor ρ. In a colliding regular remapping, the mapping index of the evicted LLA' is incremented δ times before a free PLA is found. With the same randomness assumption above, δ is geometrically distributed with parameter ρ and mean $1/\rho$. That means we need to choose ρ such that $1/\rho \ll S$, otherwise, colliding regular remappings will frequently consume the entire window of S indices, invoking a costly catch-up remapping. Sample parameters that satisfy this requirement are $\rho = 0.15$ and $S = 64 \gg 6.66$.

Performance Estimate for the 1-LLA Workload

For the 1-LLA workload and the catch-up algorithm we choose in this work (see Section 2.3.3), the last of every S consecutive regular remappings will invoke a catch-up remapping. Thus the proposed architecture can level the 1-LLA writes across the entire space of N PLAs, costing N/S catch-up remappings in total. Each catch-up remapping costs on average K/N internal-copy writes per PLA, giving on average $\frac{N}{S}\cdot\frac{K}{N}=\frac{K}{S}$ internal-copy writes per PLA in the device lifetime. When K/S is not a large fraction of w_{max}, the proposed architecture will be able to reach a high value of utilization.

Calculating the Trigger Threshold

Recall from Section 2.3.1 that ϕ is the wear threshold above which the PLA only serves remapping writes, and no direct host writes. This immediately gives the criterion by which we need to set ϕ: the remaining endurance of $w_{max}-\phi$ writes should suffice for serving all future remapping writes into this PLA. If ϕ is set too high such that this condition is not met, a remapping operation will cause a PLA to exceed w_{max} writes, ending the device lifetime prematurely before claiming all unused endurance. With this condition in mind, we want to set ϕ as high as possible to maximize the number of host writes served by each PLA.

Deriving the Optimal ϕ for the 1-LLA Workload

We denote $\alpha \triangleq \phi/w_{max}$ as the fractional threshold, and seek to find the optimal α. In the following quantitative discussion we assume the 1-LLA workload, and consider only the internal-copy writes of catch-up remappings (the other internal-copy writes, during regular remappings, are negligible in number). According to the condition on ϕ stated earlier in Section 7, setting a fractional threshold of α

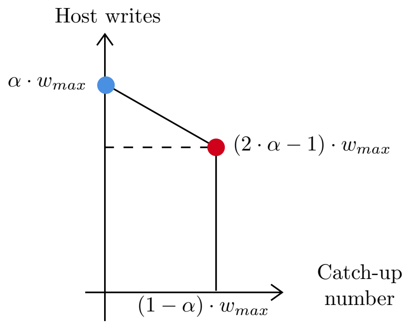


Figure 11: Host writes per PLA vs. the catch-up number.

leaves sufficient endurance for $(1-\alpha)w_{max}$ catch-up remappings. Between the t-th and $(t+1)$-th catch-up remappings, S PLAs each serve $\alpha w_{max}-t$ host writes (t writes are consumed by the previous catch-up remappings). In other words, the number of host writes per PLA is decreasing linearly as a function of the catch-up number with slope -1; this is shown graphically in Figure 11. The total number of host writes served by the device equals the area of the trapezoid in the figure, multiplied by the constant S. Thus the optimal α maximizes the expression for this area, given by

$$\frac{(\alpha w_{max}+(2\alpha-1)w_{max})\cdot(1-\alpha)w_{max}}{2}. \tag{5}$$

Taking the derivative and equating to zero, we get $\alpha^* = 2/3$. Note that this optimal α^* applies so long that $(1-\alpha^*)w_{max} \le N/S$, because the right-hand side is an upper bound on the number of catch-up remappings until utilizing the entire device. This is equivalent to the condition $N/(Sw_{max}) \ge 1/3$. For the complement case $N/(Sw_{max}) < 1/3$, the x-axis of Figure 11 can reach the maximal number of catch-up remappings N/S with $\alpha \le 1-N/(Sw_{max})$, so setting $\alpha = 1-N/(Sw_{max})$ maximizes the total number of host writes. We summarize the optimal values of α in the following equation:

$$\alpha_{\text{opt}} = \begin{cases} 1-\frac{N}{Sw_{max}}, & \frac{N}{w_{max}} < \frac{S}{3} \\ \frac{2}{3}, & \text{otherwise} \end{cases} \tag{6}$$

We use this α_{opt} to set the trigger threshold in our evaluations in Section 4. Note that the first case of (6) is the more favourable one that fully utilizes the N PLAs of the device. As we increase S, we remain in this favourable case for larger N/w_{max} ratios.

Proof of Property 2

If both i and j are in the range $[0,\ldots,N-1]$, all the non-zeros in their binary representations are confined to the $m = \log_2 N$ right-most bits of the index field. If both i, j map the same LLA to the same PLA, then both $[LLA|i|PLA]$ and $[LLA|j|PLA]$ must be codewords. When subtracting (modulo 2) these codewords, we get a third codeword all of whose non-zeros are confined to m or less consecutive coordinates, which is a contradiction when the code is cyclic with redundancy $r = m$.