Composing Lifetime Enhancing Techniques for Non-Volatile Main Memories

Andrés Amaya García
ARM Research
Cambridge, United Kingdom
andres.amayagarcia@arm.com

William Wang
ARM Research
Cambridge, United Kingdom
william.wang@arm.com

René de Jong
ARM Research
Cambridge, United Kingdom
rene.dejong@arm.com

Stephan Diestelhorst
ARM Research
Cambridge, United Kingdom
stephan.diestelhorst@arm.com

ABSTRACT

Emerging byte-addressable non-volatile memory (NVM) technologies, such as PCM and ReRAM, offer significant gains in terms of density and power consumption over their volatile counterparts. Their write endurance is, however, orders of magnitude lower than DRAM, potentially causing devices to fail in seconds. Therefore, to use NVM as DRAM replacement, writes must be managed carefully.

In this paper, we study the endurance problem for NVM main memories with realistic server workloads. We explore three existing techniques to extend NVM lifetime: last-level cache replacement policies, compression, and NVM wear-leveling. The first two approaches increase lifetime by reducing the write traffic from the cache to the main memory. Wear-leveling spreads writes and reduces hotspots responsible for fast failures.

Even though custom replacement policies and compression are common in DRAM caches inside NAND flash devices, we find that they provide insufficient lifetime gains for NVM main memories with realistic server workloads. Caching writes is effective, but adapting the replacement policy only provides modest write reductions by 10%, while compression schemes must quadruple the cache capacity to achieve reductions of 20%. In both cases, the lifetime increases by an order of magnitude, which, for example, translates in an improvement from 12 days to 6 months for PCM. In contrast, wear-leveling algorithms can increase overall lifetime by at least two orders of magnitude, for instance, from 12 days to 15 years for PCM. These results indicate that wear-leveling techniques are more promising to ensure that NVM technologies are feasible to use as DRAM replacement.

CCS CONCEPTS

• **Computer systems organization** → *Architectures*; • **Hardware** → **Communication hardware, interfaces and storage**;

KEYWORDS

NVM, Cache replacement policy, Cache compression, wear-leveling, Endurance

ACM Reference format:
Andrés Amaya García, William Wang, René de Jong, and Stephan Diestelhorst. 2017. Composing Lifetime Enhancing Techniques for Non-Volatile Main Memories. In *Proceedings of MEMSYS 2017, Alexandria, VA, USA, October 2–5, 2017,* 11 pages.
DOI: 10.1145/3132402.3132411

MEMSYS 2017, Alexandria, VA, USA

978-1-4503-5335-9/17/10…$15.00
DOI: 10.1145/3132402.3132411

1 INTRODUCTION

Non-Volatile Memory (NVM) technologies feature high density, low cost and persistence of stored information without refreshes. Furthermore, owing mainly to its non-volatility property, NVM is extremely power efficient compared to its volatile counterparts, including DRAM. So far, NVM technologies have been constrained to applications in the lower layers of the memory hierarchy inside long-term storage devices. Nevertheless, emerging memory technologies, such as Phase Changing Memory (PCM) and Resistive RAM (ReRAM), are also byte-addressable similar to DRAM. This greatly simplifies the operation of NVM-based devices, and has resulted in strong interest in using these technologies as main memories.

Despite recent improvements, emerging NVMs still suffer from low write endurance, that is the number of writes per bit cell before the cell breaks. This is one of the fundamental problems that prevented the use of previous NVM technologies, such as NAND flash, in main memories. To put that into perspective, DRAM cells can be written to about 10^{16} times, while PCM and ReRAM have write endurances in the range 10^{6}-10^{8} [13, 14]. In other words, NVM technologies have limited endurance which results in very short service lifetime making them impractical as a replacement for DRAM.

In this work, we estimate the gap in the lifetime of an NVM main memory that must be satisfied before these technologies can be used to replace DRAM. We profile the write traffic generated by four common server workloads while running on a simulated model of a typical server system, and explore the use of three technique to extend the lifetime of NVM main memories: cache replacement policies, cache compression and wear-leveling. To evaluate cache replacement policies, we implement four different schemes from the

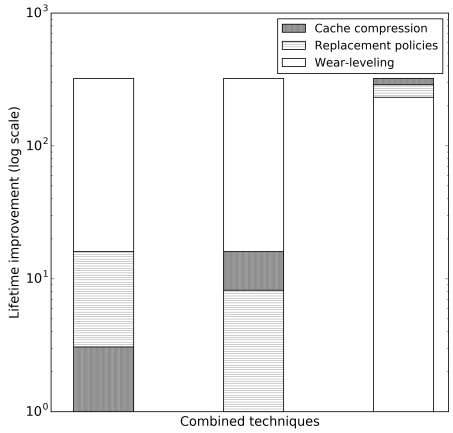

Figure 1: Average lifetime improvement achieved by progressively combining techniques analyzed in this paper. Each portion of a bar corresponds to the lifetime increase achived by a single technique. The techniques at the bottom of the stack are added first, while techniques towards the top are evaluated in conjunction with the ones below. The system used in these experiments contains L1, L2 and L3 caches of 32 KB, 256 KB and 8 MB with LRU replacement policy except the L3 when evaluating replacement policies.

literature and compare the results to the traditional Least Recently Used (LRU) policy. The motivation behind this approach is to cache the writes to reduce the number of write operations to the main memory, thus increasing the NVM lifetime. However, we find that in most cases, changing the replacement policy does not substantially reduce write operations.

The second technique compresses the cache to reduce capacity misses, and decrease the writes to main memory. We estimate that by using compression we can double the capacity of the cache without substantially increasing the physical size of the device. However, we find that increasing the cache capacity is not an efficient mechanism to achieve suitable lifetime because this only reduces the write operations by 15%.

Wear-leveling algorithms in the NVM main memory is the third and final technique we consider. We develop a simple, dynamic wear-leveling mechanism and estimate the resulting lifetime of the NVM device. We find that this scheme can increase the lifetime by one to two orders of magnitude. Furthermore, using write traffic information, we calculate the upper bound on the lifetime improvement using oracle wear-leveling to be about three orders of magnitude.

In this paper, we look at the techniques mentioned above as a black box, providing a factorial increase of lifetime (by a factorial reduction of effective writes). That way, it is possible to make no assumptions about the mechanisms in one component to assess the entire lifetime. Figure 1 summarises our results and shows that the lifetime contributions of several common techniques do not stack. Instead, while effective in isolation, their compound effect is diminished when applied in combination. Especially worrisome is the reduced effect of write avoidance techniques when coupled with wear-leveling, suggesting that write reductions obtained by plain measurements on address traces do not translate into an equivalent lifetime increase.

The aim of this research is to provide an understanding of the endurance problem and its main causes:

- We demonstrate the magnitude of the endurance problem for NVM main memories with realistic server workloads.
- We evaluate the impact of LLC cache configuration on the endurance of the NVM.
- We evaluate the effectiveness of previously proposed techniques in the context of main memory lifetime extension.
- We make recommendations based on extensive evaluation to increase the lifetime of NVM main memories.

The remaining of this paper is organized as follows. In Section 2 and Section 3, we present related work on methodologies to evaluate NVM lifetime and techniques to improve it respectively. In Section 4, we discuss the methodology used throughout this work. Detailed experimentation results for each technique are presented and analyzed in Section 5. In Section 6, directions for future work are discussed. Finally, Section 7 summarizes our conclusions.

2 RELATED WORK ON EVALUATION METHODOLOGIES

Various approaches have been used in other works to evaluate proposed write suppression and wear-leveling techniques that aim to mitigate the NVM endurance problem. In general, a set of benchmarks is selected and simulated while various pieces of data are collected depending on the technique in question. The methodologies used to evaluate write suppression techniques, such as innovative cache replacement policies, assess the impact of the idea by using almost exclusively the total number of writes to NVM compared to some baseline [11, 17, 29]. Some papers also perform weighted evaluations of their approach, since they consider that the cost of read and write operations differ [24]. This information alone does not provide enough insight into the effectiveness of the proposed solution. This is because write reduction does not necessarily translate to a significant lifetime improvement. For instance, a cache replacement policy might reduce the overall write traffic to NVM, but increase the number of writes to specific memory locations causing uneven wear out.

In contrast, research that presents wear-leveling schemes evaluates the lifetime improvement as a result of spreading the write operations evenly across the memory space. In this case the density of write operations at different granularities is used to evaluate the proposed scheme [25, 35]. To gather this information workloads are run and the frequency of write accesses is registered. The lifetime is then estimated by assuming that the NVM is accessed at that rate. This information is not enough to assess the impact that other components of the system, such as the cache, have in the performance of the wear-leveling scheme. The methodology used throughout our work collects a wide range of information from the simulated memory system to provide a complete view of the effect of any technique. We also simulate multiple techniques applied to the same system to observe how this affects the lifetime of the NVM main memory under realistic workloads.

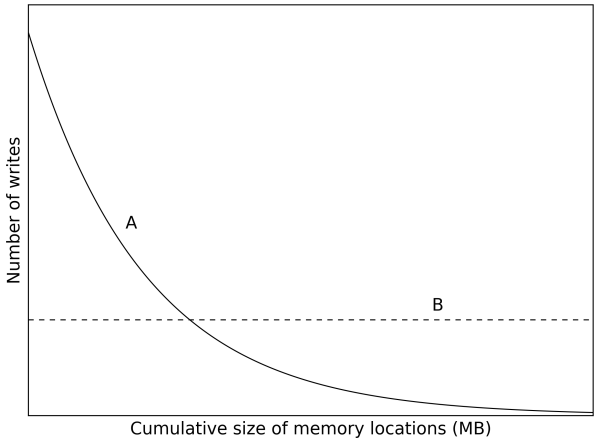

Figure 2: Example histograms displaying the number of writes per 64-byte memory location. *A* and *B* are examples of memories with poor and perfect wear-leveling respectively.

3 TECHNIQUES TO INCREASE NVM LIFETIME

The lifetime enhancing techniques discussed can be classified in two categories: write suppression and wear-leveling. The former reduces the number of writes to the NVM main memory. Cache compression and replacement policies are the two techniques we explore that fall in this category. Nevertheless, there are other ideas that can be considered as write suppression techniques. For instance, the use of DRAM caches in Non-Volatile Dual In-line Memory Modules (NVDIMM).

Wear-leveling evenly distributes the writes across the memory space to ensure that specific locations do not *wear out* relatively early compared to other locations. We say that a NVM memory cell has *worn out* when it does not reliably retain the stored state. If we consider that Figure 2 is a histogram displaying the number of writes for every memory location in the NVM, the data points in *A* are an example of poor wear-leveling. This is because very few locations that are written too many times and will malfunction much earlier than the majority of memory cells that are written seldom. In contrast, the data points in histogram *B* show an NVM that has perfect wear-leveling, since every location is written approximately the same number of times as every other location, and we expect the lifetime of *B* to be higher than that of *A*.

Next, we study how different techniques can be used as write suppression or wear-leveling strategies to enhance the lifetime of NVM main memories.

3.1 Cache Replacement Policies

In a conventional memory system, there is usually a cache hierarchy with one or more levels between the processor and the main memory which reduces the latency of memory accesses, and decreases the time that the processor is stalled. Therefore, most existing work on replacement policies concentrates on improving the cache hit ratio. We explore the use of replacement policies to reduce the write traffic from the cache hierarchy to the main memory, resulting in extended NVM lifetime.

The challenge is to find a replacement policy that reduces the write traffic without significantly hurting the hit ratio. Furthermore, the policy should be simple enough to be efficiently implemented in an on-chip cache. There are few works that propose a replacement policy for our purposes [31]. However, thanks to the widespread use of NVM storage systems, such as flash-based Solid State Drives (SSD), there are several studies on replacement policies for DRAM caches to reduce write traffic to storage [9, 11, 17–19, 24, 29, 32, 33]. There are significant differences between the NVM storage setting and ours. In the former case, a system consisting of a host computer connected to a storage device is assumed. The DRAM cache replacement policy is either implemented in a buffer inside the NVM storage device or the page cache maintained by the operating system in main memory. In contrast, we evaluate replacement schemes in an on-chip LLC, and require the policy to be operational there (usually L2 or L3).

To predict the future access pattern of a cache line, DRAM-based cache replacement policies consider valuable indicators other than recency as is the case in Least Recently Used (LRU). For example, Clean-First LRU (CFLRU) considers both recency and the clean or dirty state of a cache line, and splits the cache into predefined, fixed size regions: working and clear-first [24]. The former behaves as a normal LRU cache, while the latter is similar to a write buffer because clean cache lines are given priority for eviction. Therefore, large clean-first regions negatively impact the miss rate since fewer clean lines are kept in the cache.

LRU Write Sequence Reordering (LRU-WSR) [11] is an alternative approach that tags dirty cache lines as hot or cold. LRU-WSR gives dirty lines a "second chance" by migrating them to the Most Recently Used (MRU) position and chosing to evict cold or clean lines instead. The Cold-Clean-First LRU (CCF-LRU) [17] replacement policy attempts to improve on both LRU-WSR and CFLRU by considering the hot and cold state of both clean and dirty cache lines. The algorithm maintains a list for dirty and hot clean lines and a separate list for cold clean lines. The latter is given priority for eviction, but if it is empty, a victim will be selected from the former.

There are replacement policies that use learning rules to adjust the cache configuration according to the access patterns of the workload [20, 29]. The Adaptive Page Replacement Algorithm (APRA) splits the cache in two sectors: the first sector behaves like CFLRU, while the second, called *ghost cache*, only holds cache line tags and the dirty flag of lines evicted from the other sector. The learning rule adjusts the size of the sectors at runtime to reduce the probability of evicting dirty cache lines when data is written often without significantly impacting the miss rate [29]. Unfortunately none of these schemes were designed for on-chip caches, thus it is unclear whether their implementation is feasible in silicon. However, replacement policies of LLCs operate off the timing critical path and can use more complex replacement schemes.

3.2 Cache Compression

The high access latencies of off-chip memory have motivated the widespread use of on-chip caches. However, the constraints inherent to the memory technologies mean that microprocessor designers must compromise on the amount of memory available on-chip to enable suitable access latencies to the cache. Compression has long been discussed as an alternative mechanism to increase the effective size of the cache without actually adding memory cells [1–3, 7, 8, 10, 27, 28]. In this case, the *effective size* refers to total uncompressed size of the data stored in the cache. This idea is appealing for our purposes since it mitigates cache capacity misses that ultimately result in write operations. That is, a compression scheme could enlarge the effective capacity of the cache enough such that the write traffic from LLC to main memory is substantially reduced, ultimately increasing the lifetime of the NVM main memory.

Despite the obvious appeal, cache compression poses its own set of challenges. It is imperative that the scheme is lossless, power efficient and its implementation requires low area. Furthermore, the compression hardware must not degrade the performance of the cache, specifically those operations that are typically in the critical path of accesses to memory, such as decompression. It is also desirable that the compression ratio is as large as possible and that data can be randomly accessed at the granularity of an uncompressed cache line.

In the literature, there are various proposals for cache compression schemes. Zero-Content Augmented Cache (ZC) [8] is perhaps one of the simplest approaches. The designers of this scheme observed that on some applications over 20% of memory accesses concerning null data blocks. Therefore, ZC compresses cache lines by storing the tags and discarding the null data. Despite its simplicity, ZC achieves average compression ratios of approximately 1.5, yet its performance is largely workload dependent. Frequent Pattern Compression (FPC) [2] is a slightly more complex approach that matches the input data against a prefix table of predefined patterns. The compressed output consists of an index of the matching prefix in the table and some bytes. A similar approach, Frequent Value Encoding (FVE), [34] replaces the pattern table by a dictionary of values whose contents can be selected in advance or updated dynamically. However, FVE was originally thought as a compression scheme for data buses. The compression ratio of these approaches is reported to lie between 1.5 and 2.8 depending on the implementation details and benchmarks.

C-Pack [7] is a more refined dictionary-based scheme specifically designed for on-chip caches. The algorithm matches the input data against both statically decided patterns, and a dynamically constructed dictionary. Furthermore, a cache architecture, referred to as *pair matching*, is also proposed to store cache lines depending on their compressed ratio and that of their partner. It has been reported that C-Pack can increase the effective cache size by approximately a factor of 1.7.

There are works that also propose alternative cache compression architectures and use existing compression algorithms [1, 27, 28]. These ideas have varying levels of success, but generally, they are capable of improving the effective cache size by a factor in the range of 1.5 to 2.5.

3.3 Wear-leveling

Contrary to the techniques previously discussed, wear-leveling is a mechanism that is directly applied to the main memory itself. The idea behind this technique is to evenly distribute the write operations across the memory space to ensure that the memory cells degrade evenly. Wear-leveling is an approach often used to extend the lifetime of flash storage devices [4, 6, 12, 22]. The specifics vary depending on the scheme and its application, but generally wear-leveling is achieved by maintaining indirection tables from logical to physical addresses. Additional tables may also be maintained to track the write count of the storage blocks, and route subsequent write operations to seldomly written locations. More sophisticated algorithms, such as Rejuvenator [22], also aim to identify and regularly migrate rarely written logical addresses to free least written physical locations.

A number of wear-leveling schemes have also been proposed for PCM based main memories [14, 25, 26, 35]. Row shifting is a mechanism that reads and compares the memory block with data to be written, then it only changes the value of the memory cells that differ [35]. Since this would result in uneven number of writes across a memory row, the bits are periodically shifted. Similar approaches have been proposed at different levels of granularity. For instance, when a virtual page is written to a PCM main memory, only the cache lines that differ need to be replaced. To ensure that the device is worn-out evenly, cache lines are periodically rotated within the boundaries of the virtual page [26]. Start-Gap is a wear-leveling scheme for PCM main memories that aims to reduce the size of the indirection tables commonly used for flash wear-leveling [25]. To do so, Start-Gap uses an algebraic mapping between logical and physical addresses. The nature of the expression ensures that the memory blocks are rotated across the whole memory space.

4 METHODOLOGY

We use state-of-the-art simulation tools and explore a large number of configurations of the memory system. Data related to the caches and the traffic across the memory hierarchy is collected throughout the process and later analyzed. The remaining of this section describes our methodology in detail.

4.1 Experimental Setup

Our experiments are conducted using atomic models of server systems developed upon the gem5 toolset [5]. Gem5 is an open-source, modular platform that enables the user to simulate one or more computer systems. It contains a large number of highly configurable components that can be used to construct tailored models for computer-system architecture research. In our case, we use gem5 to simulate server workloads on 64-bit ARM systems, and collect data about the traffic across the memory hierarchy. In particular, we are interested in the performance of the LLC, as well as the volume and locations of read and write accesses from the LLC to the main memory.

We experiment with a large number of cache configurations with hierarchies of two and three levels deep and various sizes as shown in Table 1. We are mainly interested in the traffic generated by a workload across the memory system. Therefore, we strive to simplify other components to reduce the overall simulation time;

Table 1: Cache configurations for simulated systems. In all cases, the cache line size is 64-bytes.

L1 Cache	L2 Cache	L3 Cache
16 KB	512 KB	-
32 KB	1 MB	-
64 KB	2 MB	-
64 KB	4 MB	-
32 KB	256 KB	1 MB
32 KB	256 KB	2 MB
32 KB	256 KB	4 MB
32 KB	256 KB	8 MB
32 KB	256 KB	16 MB
32 KB	256 KB	32 MB
32 KB	256 KB	64 MB
32 KB	256 KB	128 MB
32 KB	256 KB	256 MB

for instance, the underlying processor selected is an atomic model of a single 64-bit ARM core. As a result, we are able to experiment with realistic workloads and a large set of memory configurations. To avoid introducing noise due to virtual page swapping, we set the size of the main memory to 16 GB, significantly larger than the memory footprint of any of the workloads.

4.2 Simulation and Offline Data

In general, our data gathering and analysis process can be divided in two parts: online and offline. The former refers to the data gathering performed while the simulation of the system is running in gem5. At this stage, we record various pieces of information, including (but not limited to) cache line miss and hit rate, dirty portion of the cache over time and cache line utilization. Furthermore, we collect the read and write memory access traces from the LLC to the main memory after the virtual address of the operation has been resolved. The trace data includes time, physical memory address, size and type (read or write) of each access. When the simulation terminates, we proceed to analyze the data offline using software tools developed specifically for this project. These tools accept the gem5 simulation output and memory access traces and transform it into a manageable format for manual inspection.

Depending on the technique under evaluation, we make adjustments to the process described above. To explore replacement policies, we implement four different schemes, CFLRU, LRU-WSR, CCF-LRU and APRA (see Section 3), in the LLC cache model in gem5, and use them to compose the models for simulation. We also conduct the experiments with the traditional LRU policy to use it as a baseline for the other replacement policies. In the case of cache compression, we compare the expected effective cache size obtained from our literature survey to the simulation results of systems simulated using LRU and various cache sizes. We are then able to estimate the impact of larger cache capacities on the lifetime of the NVM main memory. Finally, to estimate the impact of wear-leveling techniques, we simulate the accesses described in the traces to a main memory device that has some wear-leveling scheme. We significantly reduce the size of the main memory to approximately 200 MB more than the memory footprint of the workload to decrease the amount of overprovisioned memory. This stresses the wear-leveling algorithms because they rely on migrating data to free memory locations when writes occur. Throughout our evaluation, we compute the lifetime values for the different techniques for illustration purposes. Due to the nature of atomic simulation in gem5, these times may have small inaccuracies that are, however, insignificant to the orders of magnitude impacts of the effects we are analysing; and the relative factors of improvement remain correct.

4.3 Workloads

For this investigation, we have chosen the workloads Memcached [21], Server-Side Java (SSJ) [30], PageRank [16] and Forest-fire [16] running under Ubuntu 14.04 LTS. We select these server-type applications for our experiments due to their widespread use in industry. Furthermore, they represent a broad selection of tasks commonly performed by server machines, including database caching and graph handling.

Memcached is an open-source, generic, distributed memory object caching system [21]. It is mainly used to cache data loaded from external data sources thus speeding up dynamic database-driven applications. For our experiments, Memcached version 1.4.21 is loaded in a server and requests are continuously sent from an independent client machine through an Ethernet connection. The data transmitted corresponds to extracts from the popular social network Facebook. The experiment is entirely simulated on gem5, yet data is only collected for the server.

Server-Side Java (SSJ) is part of the SPECpower_ssj 2008 suite [30]. This workload is an industry-standard benchmark to evaluate performance characteristics of servers.

Finally, Forest-fire and PageRank are two graph workloads from the Stanford Network Analysis Project (SNAP) version 2.4 [16]. The former is a software that generates a graph based on the Forest-fire model [15] with fordward and backward burning probabilities of 0.3 and 0.25 respectively. Finally, PageRank is the algorithm developed by Larry Page and Sergey Brin to rank websites according to the number and quality of their links [23]. We run PageRank on the Webgraph from the 2002 Google programming context.

5 RESULTS

In this section, we present and analyze the results of our experiments in four parts. Firstly, we profile the write traffic generated by our selected server workloads and evaluate the lifetime of a system that does not have any countermeasures. The objective is to estimate the magnitude of the endurance problem and set a suitable baseline to quantify any improvements. The remaining three parts of the section evaluate each of the techniques: cache replacement policies, cache compression and wear-leveling. Furthermore, we compare the improvements achieved by the techniques with each other and draw conclusions.

5.1 Quantifying the Endurance Problem

NVM technologies have physical limitations that cause the memory cells to degrade when certain operations are performed. We say that a memory device has *worn out* when the first memory cell malfunctions, and we define *lifetime* as the interval of time between

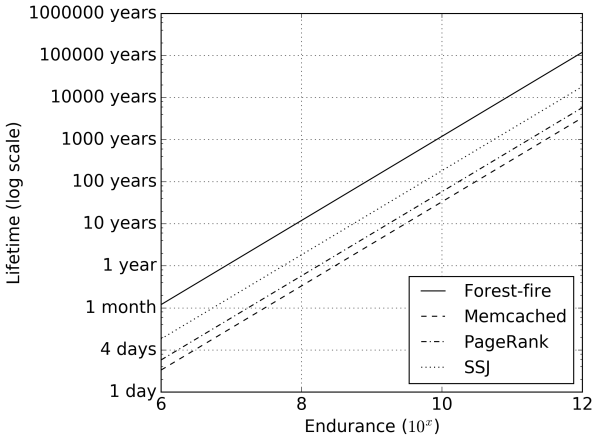

Figure 3: Lifetime of NVM main memory as endurance increases for different workloads running on a system with cache sizes 32 KB, 256 KB, 8 MB for L1, L2 and LLC respectively. The LLC replacement policy used is LRU.

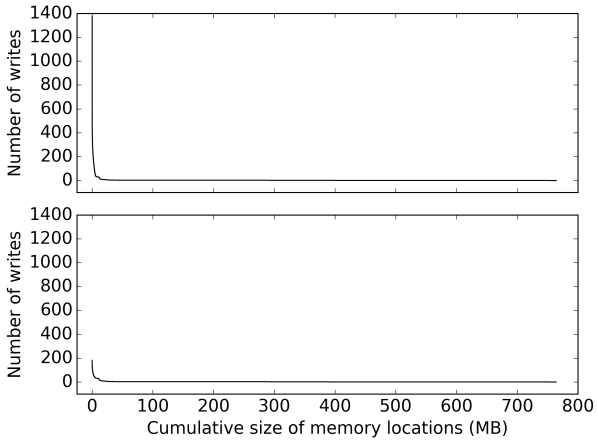

Figure 4: Number of writes per 64-byte memory location when running Memcached on a system with cache sizes 32 KB, 256 KB, 8 MB for L1, L2 and LLC respectively. The LLC replacement policies used are LRU (top) and LRU-WSR (bottom). A highly skewed graph indicates that a small portion of the memory is damaged when the memory is declared worn-out. Note the reduction to the most written location when an alternative cache replacement policy is used.

the first usage until the first malfunction. Our purpose is to provide an estimate instead of considering the specific properties of the various NVM technologies.

The expected lifetime of a NVM main memory for different endurance values is shown in Figure 3. Given that for modern NVM technologies, including PCM and ReRAM, the endurance (writes till bit cell failure) is in the range of 10^6-10^8 write operations, there is a significant lifetime gap compared to DRAM whose endurance borders 10^{16}. Nevertheless, we consider that for NVM main memories to be practical, we require their lifetime to be at least 5 years. This is because after this time, most users of server-type hardware will upgrade their equipment regardless of whether it is functional. In this sense, the endurance gap is two or three orders of magnitude. However, the information in Figure 3 can be used to calculate the endurance gap for systems with different service lifetime requirements.

5.2 Properties of Write Traffic

To characterise the properties of write traffic for our selected server workloads, we first examine the distribution of instructions executed by the software listed in Table 2. According to this information, all workloads have similar proportions of memory accesses. However, Forest-fire and Memcached are relatively write intensive workloads since the proportion of their read and write accesses is almost 1:1. This is expected since the former must constantly modify the graph that it operates on as it creates communities between the nodes. Similarly, Memcached effectively converts the main memory into a cache of database objects, which is potentially updated whenever there is a client request.

After analyzing the main memory access traces as described in Section 4, we are able to extract more detailed information regarding the location and frequency of the accesses across the memory space. We register the addresses of every cache line that is accessed. Excluding duplicates and counting the modified cache lines that are written back to main memory, we estimate the total memory footprint for each workload as displayed in Table 2. In all cases, the memory footprint of the server workloads is larger than the added cache sizes of any configuration we model by a factor of at least 2 (see Section 4). We obtain the histograms shown in Figure 4 by computing the frequency of the writes to main memory for each cache line. Despite having a large memory footprint, the write traffic generated by the workloads is skewed since less than 20% of all cache lines are written more than 5 times. According to our definition of wear out, this implies that less than a fifth of the NVM is damaged when the memory is declared worn-out, while the majority is scarcely used. One cause of such uneven behaviour is the common programming practice of storing the state information of the software in memory locations that are frequently accessed and updated as execution proceeds. For instance, in the implementation of a linked list, the most commonly accessed locations are likely to be the memory that stores the list size and the pointer to the element at the head rather than the list contents.

5.3 LRU Cache Configuration and Write Traffic

We investigate the effect of the LRU associativity level and cache size on the overall write traffic from the LLC to the NVM main memory. To evaluate the former, we simulate a three-level deep cache hierarchy with sizes 32KB, 256KB and 8 MB respectively and progressively increase the number of ways in the LLC from 16 until reaching full associativity. We observe that both the miss rate and the generated write traffic experience relatively small variations of less than 15%. Therefore, we conclude that when the associativity is increased over a threshold, evictions due to conflict

Table 2: Execution details for each server workload.

Workload	Instructions	Memory access instructions	Read accesses	Write accesses	Memory footprint
Forest-fire	14 Billion	31%	54%	46%	564 MB
Memcached	64 Billion	33%	54%	46%	882 MB
PageRank	47 Billion	31%	62%	38%	836 MB
SSJ	22 Billion	34%	65%	35%	1.1 GB

misses are reduced to a minimum and do not impact the write traffic significantly.

The second parameter of the cache configuration that we modify is the size of the LLC. In this case, we conduct experiments with a three-level cache hierarchy where the L1 and L2 sizes are fixed to 32KB and 256KB respectively. In contrast, the L3 size varies from 1 MB to 256 MB. Figure 5 illustrates the change in the write traffic from LLC to NVM main memory as the cache size increases. Memcached exhibits a rather exceptional behaviour since there is significant decrease in the write traffic as the cache size increases. In contrast, the other server workloads show a rather steady decrease. The average write traffic reduction is about 15% with each doubling of the cache size. In reality, it is difficult to envision the implementation of on-chip caches large enough to close the lifetime gap shown in Figure 3. For this reason, we consider that the improvement achieved by increasing the size of the on-chip cache alone is not significant enough to enable the usage of NVM technologies as main memory.

If we consider that in the best case, we only write each cache line to main memory exactly once, then we can think of the memory footprint as the ideal target for write traffic reduction. That is, in the best case, the total size of the data written to main memory equals the memory footprint of the workload. Figure 6 shows the ratio between these two metrics as the cache size increases for each server workload. With the exception of size 256 MB, the write traffic generated is at least twice at much as the ideal case for most workloads. This indicates that there is scope to study techniques, such as cache replacement policies, that suppress write operations from the LLC to the main memory.

To explore the causes of the rather limited reduction of write traffic, we instrument the gem5 simulator to record the percentage of the LLC that is dirty over time. We set the sampling rate at 10 million memory accesses and run the experiment once again. Figure 7 shows the results and provides some insight on the memory access patterns of these server workloads. In these graphs, an increasing trend translates in cache lines being written, while a decrease corresponds to dirty cache lines being evicted and written back to main memory. Therefore, continuous fluctuation in the graphs is undesirable because the cache lines are continuously written and evicted. From this, it can be seen that some workloads such as Memcached and SSJ have significantly better memory access patterns in the sense that it is easier for the cache to predict which data will be reused. As a result, for those workloads, we can expect that increasing the cache size could be an effective mechanism to reduce the overall write traffic. In contrast, PageRank and Forest-fire have seemingly random access patterns and many write-heavy phases. This is expected because they are graph workloads and the shape of the input data largely dictates their behaviour.

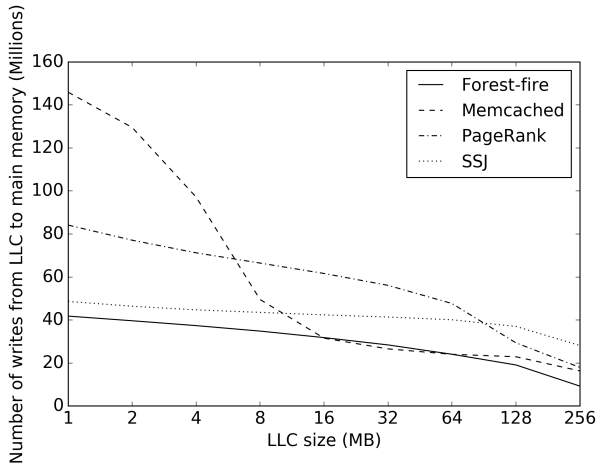

Figure 5: Effect of the cache size on write traffic from LLC to main memory. In all cases the LLC replacement policy is LRU and the L1 and L2 cache sizes are 32 KB and 256 KB respectively.

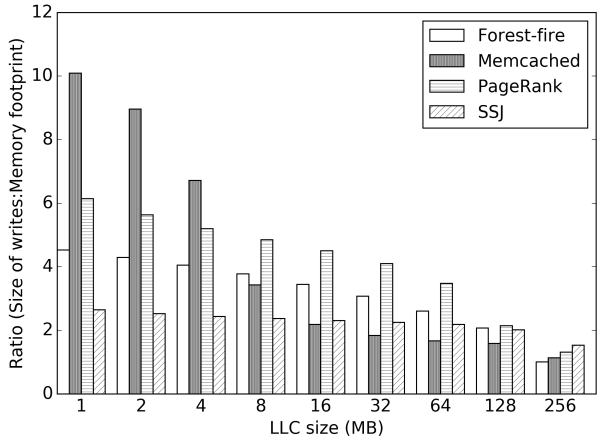

Figure 6: Ratio between memory footprint and data written to main memory as cache size increases. In all cases the LLC replacement policy is LRU and the L1 and L2 cache sizes are 32 KB and 256 KB respectively.

5.4 Cache Replacement Policies

As described in Section 4, we implement and simulate four replacement policies in gem5: CFLRU, LRU-WSR, CCF-LRU and APRA (see

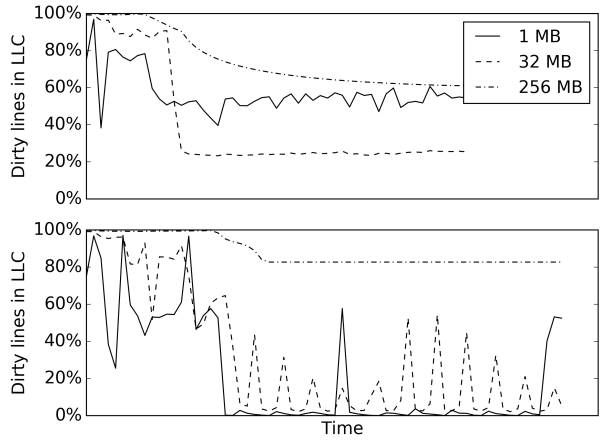

Figure 7: Percentage of the LLC that is dirty overtime for different cache sizes while running Memcached (top) and PageRank (bottom). In all cases the LLC replacement policy is LRU and the L1 and L2 cache sizes are 32 KB and 256 KB respectively.

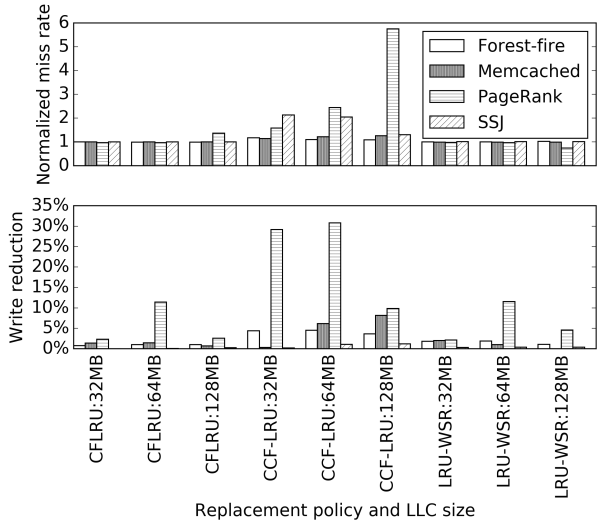

Figure 8: The impact of different cache replacement policies and LLC sizes in write traffic compared to LRU. In all cases the L1 and L2 cache sizes are 32 KB and 256 KB respectively.

Section 3). We consider that these schemes provide a representative sample of all replacement policies, enabling us to obtain an estimate of the lifetime improvement that could be achieved by using alternatives to the traditional LRU. Figure 8 shows the effect of each policy in write traffic and miss rate compared to LRU. With some exceptions, in general, the write traffic reduction is below 10%, and the miss rate remained mostly unchanged compared to LRU. This is not unexpected because all these policies are largely based on LRU with some modifications. That is, since there is little variation in the miss rate, we can assume that to some extent the data maintained in the cache by the replacement policies at any point in time is similar to the data in the LRU cache. The small traffic write reductions corresponds to the few cache lines that are frequently evicted by LRU, but not by the other policies. CCF-LRU is the only replacement policy that provides significant improvements in write traffic, especially for the heavily write-phased workloads. Nevertheless, it achieves this at the expense of a significantly worse miss rate compared to LRU as illustrated in Figure 8. The likely cause of this compromise is that some of our server workloads are relatively write-intensive. Therefore, CCF-LRU encountered a large number of hot cache lines, which were rarely evicted from the cache causing starvation of clean lines.

Comparing the distribution of writes across the memory space, it can be seen from Figure 4 that the cache replacement policies have acted as a form of wear-leveling technique. Despite the small reduction in write traffic from LLC to main memory compared to LRU, the replacement policies have retained frequently written lines longer within the cache while evicting seldomly written data. On average, the writes from LLC to main memory for the most frequently written locations has decreased by over 70%.

5.5 Cache Compression

From the literature survey presented in Section 3, we can conclude that on average we could approximately double the effective size of the cache by using some compression scheme.[1] Comparing this information with the data in Figure 5 we observe that the reduction in write traffic would be 15% on average. For instance, if we consider that the physical size of our LLC is 8 MB, with cache compression the effective size would be about 16 MB. However, to obtain a 20% reduction in write traffic, we would need an effective size of 32 MB. Since we have not found evidence that we can quadruple the cache capacity using compression, we consider that this technique on its own is not enough to enable the use of NVM main memories.

5.6 Wear-leveling

As discussed in Section 4, we run post processing tools on the memory access traces collected during simulation. In this way, we are able to estimate the performance of wear-leveling schemes. We evaluate two algorithms, we call these *perfect* and *basic* wear-leveling. The former is an ideal solution that guarantees that given any two memory locations, the difference in write count is at most one. Effectively, the lifetime of the system can be calculated using the formula

$$LifetimeInSecs = MemEndurance/WritesPerSec$$

where $MemEndurance$ is the endurance factor for the NVM technology in question and $WritesPerSec$ is the ideal number of write operations for each memory location per second. If we consider that every memory location is written before the first memory location is written once again, then $WritesPerSec$ is the number of times in a

[1] Estimating a more accurate gain requires analyzing the details of the specific scheme, implementation and target workloads. However, our intention is to estimate the write traffic reduction from LLC to NVM main memory rather than selecting a specific compression scheme. Thus, this value is enough for our purposes.

second that any memory cell is written. This wear-leveling scheme is impractical because it assumes that on every write there is a free location whose write count is smaller than that of other locations. However, we consider this scheme to be the upper bound on life improvement that can be achieved using wear-leveling techniques.

The basic algorithm dynamically changes the physical location of the data whenever it is written. It maintains an indirection table that maps logical to physical addresses, and tracks the locations that are empty. The data movements are illustrated in Figure 10. Whenever there is a write to some logical address, the previous mapping for that address in the indirection table is invalidated and the data is stored in the next free location. Since the memory space is 200 MB larger than the memory footprint, the algorithm only has this small amount of overprovisioned memory to migrate data on write operations. For simplicity, the granularity of the data migrations is 64 bytes, the same size as an LLC cache line. Note that in our experiments, the memory size is fixed. Increasing the memorycapacity is a technique commonly known as overprovisioning, and it is commonly used in conjunction with wear-leveling to increase the NVM endurance. However, overprovisioning is outside the scope of this study.

The results of our experiments are illustrated in Figure 9. The plot shows three sets of data: the perfect wear-leveling with the best endurance values as expected; the results of the basic wear-leveling algorithm; and finally, the absence of any wear-leveling scheme. The data shows that the perfect wear-leveling scheme achieves improvements between two and three orders of magnitude. Furthermore, the lifetime increase factor is reduced as the cache size increases because the working dataset can be mostly cached. Despite the constrained main memory size, the basic algorithm improves the lifetime by one to two orders of magnitude over a system with no wear-leveling depending on the LLC replacement policy.

Another observation in Figure 9 is that the difference between the basic and the perfect algorithms is about one order of magnitude in lifetime. The performance of the basic scheme is not greater because of sparsely written (static) memory locations are not migrated often. That is, if we classify memory locations as shown in Figure 10, then locations that are repeatedly written are frequently remapped to free physical addresses. However, the static locations are migrated seldomly and prevent the wear-leveling algorithm from degrading those memory cells as fast as the rest of the memory. Identifying static locations is complex and can also result in worse lifetime if data is migrated too often.

Figure 9 also shows the lifetime resulting from the use of the cache replacement policies in Section 3. Whenever wear-leveling is absent, the lifetime while using alternative replacement policies is superior to simply using LRU by an order of magnitude. This occurs because the LLC retains the most frequently written lines longer than an LRU cache, acting as a wear-leveling mechanism (see Figure 4). If wear-leveling is used, the difference in performance between replacement policies is negligible, so only the lifetime of a system using LRU is displayed in Figure 9. This occurs because the write traffic is mainly what directly impacts the result of wear-leveling rather than the specific memory locations written.

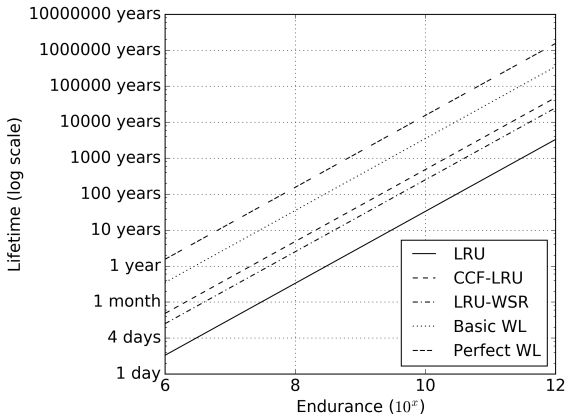

Figure 9: Lifetime of NVM main memory as endurance increases while using different wear-leveling algorithms and LLC replacement policies. Memcached running on a system with cache sizes 32 KB, 256 KB, 8 MB for L1, L2 and LLC respectively.

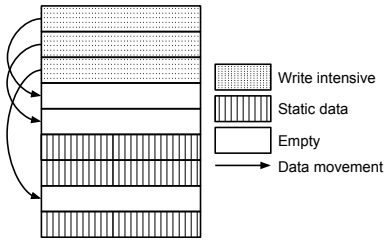

Figure 10: Classification of memory locations as seen by the wear-leveling algorithm and direction of data movement on write operations.

5.7 Combining Techniques

Figure 1 shows the improvements that result from combining multiple techniques into the same system. Wear-leveling increases lifetime by at least two orders of magnitude on average, and is the technique that contributes the most. Furthermore, it can also be seen that if wear-leveling is present, the effect of the other two techniques is minimal. In the absence of wear-leveling, the cache replacement policies increase lifetime by about one order of magnitude, approximately twice as much as cache compression.

6 FUTURE WORK

In future work we will focus on exploring wear-leveling algorithms closer to the perfect scheme. DRAM caches in NVDIMMs can be used to capture writes instead of directly using the NVM, and replacement policies and cache compression schemes can be explored with DRAM caches. Their sizes will be significantly larger than those of realistic LLCs, and the policies may therefore have a larger impact. DRAM caches may also be managed by the operating system or the applications directly, and we want to investigate the

opportunity for more complex and application-specific policies at the cost of larger (page-sized) granularities.

7 CONCLUSION

High density, low power consumption and cost are some of the features that present NVM technologies as a promising candidate for future main memory devices. However, NVM also introduces serious practical challenges due to its low write endurance. Our experiments show that there is an endurance gap of two to three orders of magnitude to ensure a lifetime greater than five years for technologies such as PCM and ReRAM. To understand the causes of the problem, we profile the write traffic from the LLC to main memory generated by server workloads. The data shows that in all cases the memory footprint is significantly larger than the size of the on-chip caches. Furthermore, irregular access patterns of graph workloads, such as PageRank and Forest-fire, make it harder to suppress writes with the limited information available to the cache. We also find that the distribution of write accesses is uneven across the memory space, despite filtering though a typical cache hierarchy; very few locations are written often while the vast majority is rarely written.

We investigate the use of three techniques to increase the NVM lifetime. Firstly, we implement cache replacement policies in the LLC different from the traditional LRU. The objective is to reduce the number of write operations from LLC to main memory without negatively impacting the hit rate. Through experimentation, we find that on average the reduction in write traffic is about 10% compared to using LRU. Nonetheless, these replacement policies decrease the difference between the most and least written memory locations, which results in an improved lifetime of approximately an order of magnitude. That is, for a PCM main memory with endurance 10^7, the lifetime increases from 12 days to 6 months for Memcached.

The second technique we investigate is compression as a means to increase the effective cache size and decrease writes to NVM resulting from evictions due to capacity misses. We estimate that using compression we could double the effective cache size, yet we find that this achieves a modest write reduction by 15%. This translates to a lifetime improvement of less than an order of magnitude, which, for instance, increases the lifetime of PCM from 12 days to 2 months. This is insufficient to guarantee a reasonable service life, let alone the added delay in the data path for compression and decompression.

Finally, we experiment with wear-leveling schemes applied to the main memory itself to evenly distribute the writes across the full memory space and avoid wearing out a small subset of the device. We experiment with a dynamic algorithm and achieve an improvement of up to two orders of magnitude over a system without any countermeasures to extend lifetime. Furthermore, we find that the main obstacle to mitigate the endurance problem with this approach is the static locations that are seldomly written by the workloads. This is because if the data is rarely written back, that location will not be worn out, and the algorithm has less space to manoeuvre. We estimate that the theoretical best case lifetime improvement that could be achieved with wear-leveling is about three orders of magnitude. For illustration, wear-leveling could increase the lifetime from 12 days to 15 years for a PCM main memory with endurance 10^7. We conclude that wear-leveling techniques are essential to ensure reasonable service lifetimes for NVM main memories.

REFERENCES

[1] Alaa R Alameldeen and David A Wood. 2004. Adaptive cache compression for high-performance processors. In *Computer Architecture, 2004. Proceedings. 31st Annual International Symposium on*. IEEE, 212–223.

[2] Alaa R Alameldeen and David A Wood. 2004. Frequent pattern compression: A significance-based compression scheme for L2 caches. *Dept. Comp. Scie., Univ. Wisconsin-Madison, Tech. Rep* 1500 (2004).

[3] Angelos Arelakis and Per Stenstrom. 2014. SC2: a statistical compression cache scheme. In *ACM SIGARCH Computer Architecture News*, Vol. 42. IEEE Press, 145–156.

[4] Avraham Ben-Aroya and Sivan Toledo. 2006. Competitive analysis of flash-memory algorithms. In *European Symposium on Algorithms*. Springer, 100–111.

[5] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, and others. 2011. The gem5 simulator. *ACM SIGARCH Computer Architecture News* 39, 2 (2011), 1–7.

[6] Yuan-Hao Chang, Jen-Wei Hsieh, and Tei-Wei Kuo. 2007. Endurance enhancement of flash-memory storage systems: an efficient static wear leveling design. In *Proceedings of the 44th annual Design Automation Conference*. ACM, 212–217.

[7] Xi Chen, Lei Yang, Robert P Dick, Li Shang, and Haris Lekatsas. 2010. C-pack: A high-performance microprocessor cache compression algorithm. *IEEE transactions on very large scale integration (VLSI) systems* 18, 8 (2010), 1196–1208.

[8] Julien Dusser, Thomas Piquet, and André Seznec. 2009. Zero-content augmented caches. In *Proceedings of the 23rd international conference on Supercomputing*. ACM, 46–55.

[9] Ziqi Fan, David HC Du, and Doug Voigt. 2014. H-ARC: A non-volatile memory based cache policy for solid state drives. In *2014 30th Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 1–11.

[10] Erik G Hallnor and Steven K Reinhardt. 2004. A compressed memory hierarchy using an indirect index cache. In *Proceedings of the 3rd Workshop on Memory Performance Issues: in conjunction with the 31st International Symposium on Computer Architecture*. ACM, 9–15.

[11] Hoyoung Jung, Hyoki Shim, Sungmin Park, Sooyong Kang, and Jaehyuk Cha. 2008. LRU-WSR: integration of LRU and writes sequence reordering for flash memory. *IEEE Transactions on Consumer Electronics* 54, 3 (2008), 1215–1223.

[12] Taeho Kgil, David Roberts, and Trevor Mudge. 2008. Improving NAND flash based disk caches. In *Computer Architecture, 2008. ISCA'08. 35th International Symposium on*. IEEE, 327–338.

[13] Mark H Kryder and Chang Soo Kim. 2009. After hard drives—what comes next? *IEEE Transactions on Magnetics* 45, 10 (2009), 3406–3413.

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

[15] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2007. Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 1, 1 (2007), 2.

[16] Jure Leskovec and Rok Sosič. 2016. SNAP: A General-Purpose Network Analysis and Graph-Mining Library. *ACM Transactions on Intelligent Systems and Technology (TIST)* 8, 1 (2016), 1.

[17] Zhi Li, Peiquan Jin, Xuan Su, Kai Cui, and Lihua Yue. 2009. CCF-LRU: a new buffer replacement algorithm for flash memory. *IEEE Transactions on Consumer Electronics* 55, 3 (2009), 1351–1359.

[18] Mingwei Lin, Shuyu Chen, and Guiping Wang. 2012. Greedy page replacement algorithm for flash-aware swap system. *IEEE Transactions on Consumer Electronics* 58, 2 (2012), 435–440.

[19] Mingwei Lin, Shuyu Chen, and Zhen Zhou. 2013. An efficient page replacement algorithm for NAND flash memory. *IEEE Transactions on Consumer Electronics* 59, 4 (2013), 779–785.

[20] Nimrod Megiddo and Dharmendra S Modha. 2003. ARC: A Self-Tuning, Low Overhead Replacement Cache.. In *FAST*, Vol. 3. 115–130.

[21] Memcached. 2015. Memcached. (2015). https://memcached.org

[22] Muthukumar Murugan and David HC Du. 2011. Rejuvenator: A static wear leveling algorithm for NAND flash memory with minimized overhead. In *2011 IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 1–12.

[23] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. The PageRank citation ranking: bringing order to the web. (1999).

[24] Seon-yeong Park, Dawoon Jung, Jeong-uk Kang, Jin-soo Kim, and Joonwon Lee. 2006. CFLRU: A Replacement Algorithm for Flash Memory. In *Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES '06)*. ACM, New York, NY, USA, 234–241. DOI: http://dx.doi.org/10.1145/1176760.1176789

[25] Moinuddin K Qureshi, John Karidis, Michele Franceschini, Vijayalakshmi Srinivasan, Luis Lastras, and Bulent Abali. 2009. Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 14–23.

[26] Moinuddin K Qureshi, Vijayalakshmi Srinivasan, and Jude A Rivers. 2009. Scalable high performance main memory system using phase-change memory technology. *ACM SIGARCH Computer Architecture News* 37, 3 (2009), 24–33.

[27] Somayeh Sardashti, André Seznec, and David A Wood. 2014. Skewed compressed caches. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 331–342.

[28] Somayeh Sardashti and David A Wood. 2013. Decoupled compressed cache: exploiting spatial locality for energy-optimized compressed caching. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 62–73.

[29] Baichuan Shen, Xin Jin, Yong Ho Song, and Sang Sun Lee. 2009. APRA: adaptive page replacement algorithm for NAND flash memory storages. In *Computer Science-Technology and Applications, 2009. IFCSTA'09. International Forum on*, Vol. 1. IEEE, 11–14.

[30] Standard Performance Evaluation Corporation (SPEC). 2008. SPECpower_ssj 2008. (2008). https://www.spec.org/power_ssj2008

[31] Zhe Wang, Shuchang Shan, Ting Cao, Junli Gu, Yi Xu, Shuai Mu, Yuan Xie, and Daniel A Jiménez. 2013. WADE: Writeback-aware dynamic cache management for NVM-based main memory system. *ACM Transactions on Architecture and Code Optimization (TACO)* 10, 4 (2013), 51.

[32] Guangxia Xu, Fuyi Lin, and Yunpeng Xiao. 2014. CLRU: a new page replacement algorithm for NAND flash-based consumer electronics. *IEEE Transactions on Consumer Electronics* 60, 1 (2014), 38–44.

[33] Guangxia Xu, Lingling Ren, and Yanbing Liu. 2014. Flash-Aware Page Replacement Algorithm. *Mathematical Problems in Engineering* 2014 (2014).

[34] Jun Yang, Rajiv Gupta, and Chuanjun Zhang. 2004. Frequent value encoding for low power data buses. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 9, 3 (2004), 354–384.

[35] Ping Zhou, Bo Zhao, Jun Yang, and Youtao Zhang. 2009. A durable and energy efficient main memory using phase change memory technology. In *ACM SIGARCH computer architecture news*, Vol. 37. ACM, 14–23.