i^{2}WAP: Improving Non-Volatile Cache Lifetime by Reducing Inter- and Intra-Set Write Variations

Jue Wang[1], Xiangyu Dong[2], Yuan Xie[1,3], Norman P. Jouppi[4]
[1]Pennsylvania State University, [2]Qualcomm Technology, Inc., [3]AMD Research, [4]Hewlett-Packard Labs
[1]{jzw175,yuanxie}@cse.psu.edu, [2]xiangyud@qti.qualcomm.com, [3]yuan.xie@amd.com, [4]norm.jouppi@hp.com

Abstract

Modern computers require large on-chip caches, but the scalability of traditional SRAM and eDRAM caches is constrained by leakage and cell density. Emerging non-volatile memory (NVM) is a promising alternative to build large on-chip caches. However, limited write endurance is a common problem for non-volatile memory technologies. In addition, today's cache management might result in unbalanced write traffic to cache blocks causing heavily-written cache blocks to fail much earlier than others. Unfortunately, existing wear-leveling techniques for NVM-based main memories cannot be simply applied to NVM-based on-chip caches because cache writes have intra-set variations as well as inter-set variations. To solve this problem, we propose i^{2}WAP, a new cache management policy that can reduce both inter- and intra-set write variations. i^{2}WAP has two features: (1) Swap-Shift*, an enhancement based on previous main memory wear-leveling to reduce cache inter-set write variations; (2)* Probabilistic Set Line Flush*, a novel technique to reduce cache intra-set write variations. Implementing i^{2}WAP only needs two global counters and two global registers. By adopting i^{2}WAP, we can improve the lifetime of on-chip non-volatile caches by 75% on average and up to 224%.*

1 Introduction

SRAM and embedded DRAM (eDRAM) are commonly used for on-chip cache designs in modern microprocessors. However, the scaling of SRAM and eDRAM is increasingly constrained by technology limitations such as leakage power and cell density. Recently, some emerging non-volatile memory technologies, such as *Phase-Change RAM (PCM or PCRAM)*, *Spin-Torque Transfer RAM (STTRAM or MRAM)*, and *Resistive RAM (ReRAM)*, have been explored as alternative memory technologies. Compared to SRAM and eDRAM, these non-volatile memory technologies have advantages of high density, low standby power, better scalability, and non-volatility. For example, among many ReRAM prototype demonstrations [11, 14, 21–25], a 4Mb ReRAM macro [21] can achieve a cell size of $9.5F^2$ (15X denser than SRAM) and a random read/write latency of $7.2ns$ (comparable to SRAM caches with similar capacity). Although its write access energy is usually on the order of $10pJ$ per bit (10X of SRAM write energy, 5X of DRAM), the actual energy saving comes from its non-volatility property. Non-volatility can eliminate the standby leakage energy, which can be as high as 80% of the total energy consumption for an SRAM L2 cache [10].

Given the potential energy and cost saving opportunities (via reducing cell size) from the adoption of non-volatile technologies, replacing SRAM and eDRAM with them can be attractive. Consistent with this expectation, such a technology shift is happening starting from lower memory hierarchy levels such as storage and main memory. For example, PCM-based storage [1, 4, 5] and main memory [8, 13, 15–17, 19, 20, 26, 27] have already been explored.

However, adoption of non-volatile memory at the on-chip cache level has a write endurance issue. For example, PCM is not suitable for on-chip caches because it is only expected to sustain 10^8 writes before experiencing frequent stuck-at-1 or stuck-at-0 errors [8, 15, 17, 19, 20, 26]. For ReRAM, the write endurance bar is much improved but is still around 10^{11} [11]. For STTRAM, although a prediction of up to 10^{15} write cycles is often cited, the best endurance test result for STTRAM devices so far is less than 4×10^{12} cycles [6]. Although the absolute write endurance values for ReRAM and STTRAM seem sufficiently high for use in on-chip L2 or L3 caches, the actual problem is that the current cache management policies are not write variation-aware. These polices were originally designed for SRAM caches and result in significant non-uniformity in terms of writing to cache blocks, which would cause heavily-written non-volatile cache blocks to fail much earlier than most other blocks.

Many wear-leveling techniques have been proposed to extend the lifetime of non-volatile main memories [15, 20, 27], but the difference between cache and main memory

978-1-4673-5587-2/13/$31.00 ©2013 IEEE

operational mechanisms makes the existing wear-leveling techniques for non-volatile main memories inadequate for non-volatile caches. This is because writes to caches have *intra-set variations* in addition to *inter-set variations* while writes to main memories only have *inter-set variations*. According to our analysis, intra-set variations can be comparable to inter-set variations for some workloads. This presents a new challenge in designing wear-leveling techniques for non-volatile caches.

To minimize both inter- and intra-set write variations, we introduce i^{2}WAP (inter/intra-set Write variation-Aware cache Policy), a simple but effective wear-leveling policy for non-volatile caches. i^{2}WAP features two schemes: 1) *Swap-Shift* is enhanced from the existing main memory wear-leveling techniques and aims to reduce the cache inter-set write variation; 2) *Probabilistic Set Line Flush* is designed to alleviate the cache intra-set write variation, which is a severe problem for non-volatile caches and has not been addressed before.

Our experimental results show i^{2}WAP can effectively reduce the total write variation by 16X on average, and the overall lifetime improvement of non-volatile L2 caches is 75% (up to 224%) over conventional cache management policies. i^{2}WAP has small performance overhead (0.25% on average) and negligible hardware overhead requiring only two extra global counters and two global registers.

2 Background

2.1 Benefits of Non-Volatile Caches

It is beneficial to use non-volatile memory technologies as on-chip caches. First, compared to SRAM and eDRAM, non-volatile memory can provide denser caches due to its smaller cell. For example, ReRAM [21] was demonstrated to be 15 times denser but with a similar read speed compared to SRAM. Thus, using non-volatile memory can enable significantly larger caches with correspondingly reduced cache miss rate, improving performance over SRAM-based caches. Second, non-volatile caches can reduce energy consumptions. Previous studies have shown that caches may consume up to 50% of a microprocessor's energy [18], and leakage energy can be as much as 80% of the total cache energy consumption [10]. Using non-volatile memories can eliminate leakage energy when they are in standby, hence reducing the total energy consumption.

2.2 Limited Write Endurance

Write endurance is defined as the number of times a memory cell can be overwritten, and emerging non-volatile memories commonly have a limited write endurance. The ITRS [7] projects that the average PCM write endurance is in a range between 10^7 and 10^8. Recent ReRAM prototypes demonstrate the best write endurance ranging from 10^{10} [14] to 10^{11} [11]. The best endurance test on STTRAM devices so far is less than 4×10^{12} cycles [6].

The write endurance of some non-volatile memories may seem sufficiently high for use in L2 or L3 caches, but the actual problem is the write variation brought by state-of-the-art cache replacement policies, which would cause heavily-written cache blocks to fail much earlier than their expected lifetime. Thus, eliminating cache write variation is one of the most critical problems that must be addressed before non-volatile memories can be used to build practical and reliable on-chip caches.

3 Inter-Set and Intra-Set Write Variations

Write variation is a significant concern in designing any cache/memory subsystems with a limited write endurance. Large write variation can greatly degrade the product lifetime because only a small subset of memory cells that experience the worst-case write traffic can result in an entire dead cache/memory subsystem even when the majority of cells are far from wear-out.

While the write variation in non-volatile main memories has been widely studied [13, 15, 16, 20, 27], to our knowledge, the write variation in non-volatile caches has not. Wear-leveling in caches brings extra challenges since there are write count variations inside every cache set (i.e. **intra-set variations**) as well as across different cache sets (i.e. **inter-set variations**). In order to demonstrate how severe the problem is for non-volatile caches, we first do a quick experiment.

3.1 Definition

The objective of cache wear-leveling is to reduce write variations and make write traffic uniform. To quantify the cache write variation, we first define the *coefficient of inter-set variations* (InterV) and the *coefficient of intra-set variations* (IntraV) as follows,

$$InterV = \frac{1}{W_{aver}} \sqrt{\frac{\sum_{i=1}^{N} \left(\sum_{j=1}^{M} w_{i,j}/M - W_{aver} \right)^2}{N-1}} \quad (1)$$

$$IntraV = \frac{1}{W_{aver} \cdot N} \sum_{i=1}^{N} \sqrt{\frac{\sum_{j=1}^{M} \left(w_{i,j} - \sum_{j=1}^{M} w_{i,j}/M \right)^2}{M-1}} \quad (2)$$

where $w_{i,j}$ is the write count of the cache line located at set i and way j, W_{aver} is the average write count defined as:

$$W_{aver} = \frac{\sum_{i=1}^{N} \sum_{j=1}^{M} w_{i,j}}{NM} \quad (3)$$

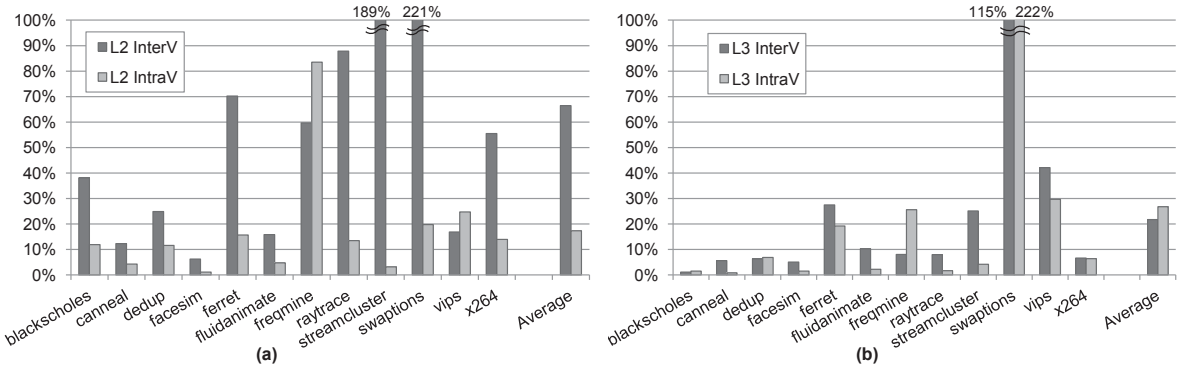

Figure 1. The coefficient of variation for inter-set and intra-set write count of L2 and L3 caches in a simulated 4-core system with 32KB I-L1, 32KB D-L1, 1MB L2, and 8MB L3 caches.

N is the total number of cache sets, and M is the number of cache ways in one set. If every $w_{i,j}$ has the same value, then $InterV = IntraV = 0$. In short, $InterV$ is defined as the CoV (coefficient of variation) of the average write count within cache sets, and $IntraV$ is defined as the average of the CoV of the write counts cross a cache set[1].

Figure 1 shows the experimental results of $InterV$ and $IntraV$ in our simulated 4-core system with 32KB I-L1, 32KB D-L1, 1MB L2, and 8MB L3 caches. The detailed simulation methodology and the setting are described in Section 8. Each graph compares inter-set and intra-set variations in L2 and L3 caches since we assume that emerging non-volatile memories will be first used in low-level caches. From Figure 1, we have two observations:

- *Large inter-set write variations:* The cache lines in different sets can experience totally different write frequencies because applications can have biased address residency. For instance, *streamcluster* has 189% $InterV$ in L2, and *swaptions* has 115% $InterV$ in L3. On average, $InterV$ is 66% in L2 and 22% in L3.
- *Large intra-set write variations:* If just one cache line in a set is frequently visited by cache write hits, it will absorb a large number of cache writes, and thus the write accesses may be unevenly distributed to the remaining M-1 lines in the set (for an M-way associative cache). The examples are: *freqmine*, which has 84% $IntraV$ in L2; *swaptions*, which has 222% $IntraV$ in L3. On average, $IntraV$ is 17% in L2 and 27% in L3.

We should also notice that write variations in L2 and L3 are greatly different. On average, inter-set variations in L3 caches are smaller than those in L2 caches. This is because L2 caches are private for each processor but the L3 cache is shared by all cores. The write variance is reduced in L3 since it mixes different requests. However, it is worth noticing that compared to an L3 cache, L2 caches have a larger average write count on each cache line because they are in a higher level of the memory hierarchy and have smaller capacity. Thus, the higher variation of L2 caches makes their limited endurance a more severe problem.

In addition, our results show the intra-set variation is roughly the same or even larger compared to the inter-set variation for some workloads. Combining these two types of write variations together significantly shortens the lifetime of non-volatile caches.

[1]We use average CoV instead of maximum CoV to keep the definitions of intra-set (the CoV of the averages) and inter-set variations (the average of the CoVs) symmetric.

4 Cache Lifetime Metrics

Cache lifetime can be defined in two different ways: **raw lifetime** and **error-tolerant lifetime**. The raw lifetime is defined by the first failure of a cache line. This lifetime is called raw because no error recovery effort is considered. On the other hand, the raw lifetime can be extended by using error correction techniques and paying overhead in either memory performance or memory capacity [8, 17, 19, 26], resulting in an error-tolerant lifetime. In this work, we mainly focus on how to improve the raw lifetime because it is the base of the error-tolerant lifetime. A further discussion on error-tolerant lifetime is in Section 9.3.

Since the raw lifetime of non-volatile caches is determined by the first memory cell that wears out, it is necessary to consider three factors: the average write count, the inter-set write count variation, and the intra-set write count variation. To estimate the overall improvement (or degradation) of the cache lifetime, we introduce a fairness metric, *Lifetime Improvement* (LI).

The target of maximizing the cache lifetime is equivalent to minimizing the worst-case write count to a cache line. Although it is impractical to obtain the worst-case write count throughout the whole product lifetime spanning several years, in this work we sample the statistics of the cache write access behavior during a short period of time by simulation and then predict write variations across the entire product lifespan. Our methodology is:

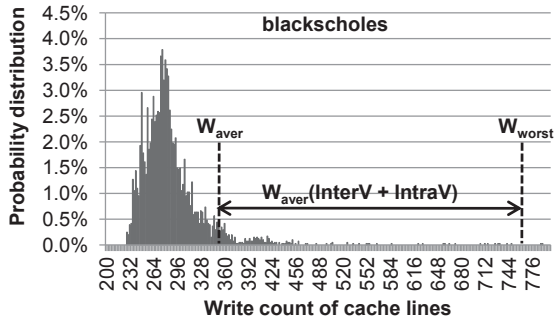

Figure 2. The L2 cache write count probability distribution function (PDF) of blackscholes.

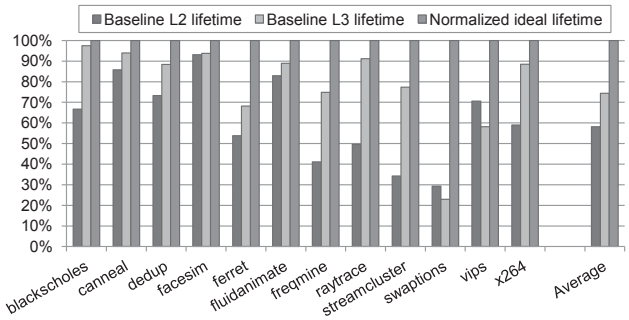

Figure 3. The baseline lifetime of L2 and L3 caches normalized to the ideal lifetime (No write variations in the ideal case).

- First, the cache behavior is simulated during a short period of time t_{sim} (e.g. 10 billion instructions on a 3GHz CPU).
- Second, each cache line write count is collected to get a average write count W_{aver}. Also, we calculate $InterV$ and $IntraV$ according to Eqn. 1 and Eqn. 2.
- Third, assuming the total write variation of a cache line is the summation of its inter- and intra-set variations[2], we then have $W_{var} = W_{aver} \cdot (InterV + IntraV)$.
- Finally, the worst-case write count is predicted as $W_{aver} + W_{var}$ to make sure to cover the vast majority of cases. While this approach is approximate, Figure 2 validates the feasibility of this model.

Assuming the general characteristics of cache write operations for one application do not change with time[3], the lifetime of the system can be defined as:

$$t_{total} = \frac{W_{max} \cdot t_{sim}}{W_{aver} + W_{var}} = \frac{W_{max} \cdot t_{sim}}{W_{aver}(1 + InterV + IntraV)} \quad (4)$$

[2]Although $InterV$ and $IntraV$ are not independent, the worst variation is the sum of these two values.

[3]If system runs different applications over time, the cache write variance can be reduced. However, in this work, we only consider the worst case. It occurs in some practical cases, such as embedded applications in which the data layout could largely remain the same.

Thus, the lifetime improvement (LI) of a cache wear-leveling technique can be expressed as:

$$LI = \frac{W_{aver\_base}(1 + InterV_{base} + IntraV_{base})}{W_{aver\_imp}(1 + InterV_{imp} + IntraV_{imp})} - 1 \quad (5)$$

The objective of cache wear-leveling is to increase LI. It needs to reduce inter-set and intra-set variations, while not significantly increasing the average write count.

Figure 3 shows the comparison between the baseline and the ideal lifetime of L2 and L3 caches, in which there are no write variations in the ideal case. It can be seen that for some workloads, such as $swaptions$, the baseline cache lifetime is only about 20%-30% of the ideal lifetime. This means that many memory cells are still healthy even when the system fails due to some cells wearing out because of the large write variation. Thus, designing a write variation-aware cache policy is critical to the future success of non-volatile on-chip caches, and we need a cache policy that can tackle both inter-set and intra-set write variations.

5 Starting from Inter-Set Write Variations

5.1 Challenges of Inter-Set Wear-Leveling

The existing wear-leveling techniques [15, 16, 20, 27] are focused on increasing the lifetime of non-volatile main memory. The key principle behind these techniques is to introduce an address re-mapping layer. This remains the same for cache inter-set wear leveling. However, there are some differences between designing wear-leveling policies for non-volatile main memory and caches.

Using Data Movement: Main memory wear leveling techniques usually use data movement to implement the address re-mapping. This is because in main memory the data cannot be lost and must be moved to a new position after re-mapping. But, data movement operations always incur overhead. First, it needs temporary data storage to move the data. Second, one cache set movement involves several reads and writes. The cache port is blocked during the data movement and the system performance is degraded. *Start-Gap* [15] is a recently proposed technique for main memory wear-leveling. If *Start-Gap* is directly extended to handle the cache wear-leveling, it falls into this catagory.

Using Data Invalidation: Another option to implement set address re-mapping for non-volatile caches is data invalidation. Cache line invalidations can be used because data in cache can be restored later from lower-level memories if it is not dirty. This special feature of caches provides a new opportunity for designing cache inter-set wear leveling techniques.

Compared to data movement, data invalidation has no area overhead. In addition, to qualify the performance overhead of these two different choices, we design and

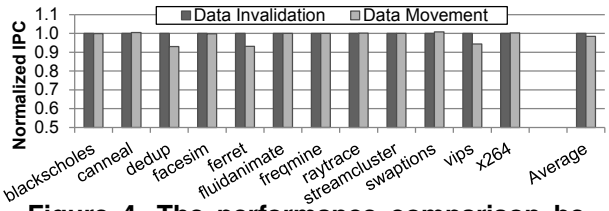

Figure 4. The performance comparison between data invalidation and data movement.

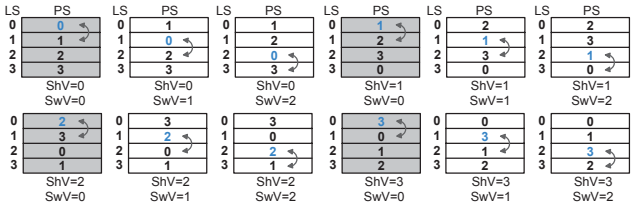

Figure 5. One shift rotation of SwS in a cache with 4 sets.

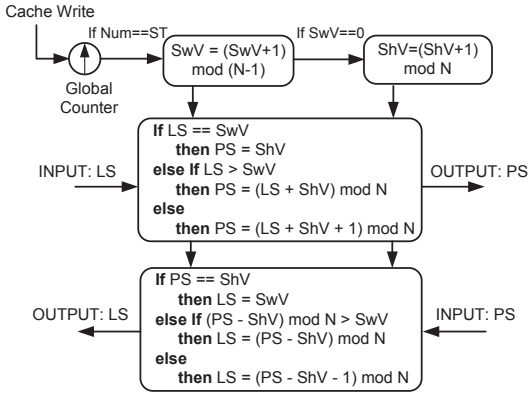

Figure 6. The mapping between logical (LS) and physical set index (PS) in SwS.

simulate two systems (see Section 8 for detailed simulation settings). In the data movement system, the data in one cache set is moved to another after every 100 writes to the cache, which is extended from the Start-gap technique [15]. On the other hand, the data is not moved but invalidated in the second system. Figure 4 shows the normalized performance of these two systems. Compared to the data movement system, the data invalidation system degrades the system performance only by 2% on average and up to 7%.

For the data invalidation system, the performance overhead comes from writing back the dirty data in the cache set to main memory and restoring data from lower level memories which should be hit later. But, it should be noticed that the latency of writing back and restoring data can often be hidden using write back buffer and MSHR techniques [12]. On the other hand, the time overhead of the data movement system cannot be optimized without adding hardware complexity since the data block movement in one cache needs to be done serially.

Therefore, as the first step of our work, we modify the previous main memory wear leveling technique and enhance it by the *Swap-Shift (SwS)* scheme to reduce the inter-set write variation in non-volatile caches.

5.2 Swap-Shift (SwS)

In contrast to the existing wear-leveling techniques for non-volatile main memory, our SwS scheme is designed for non-volatile caches. It uses data invalidation instead of data movement during cache set address re-mapping to minimize both area and performance overhead.

SwS Architecture: The goal of the SwS scheme is to shift the mapping of cache physical sets to rotate the stored data between sets. However, shifting all cache sets at once brings a significant performance overhead. To solve this problem, SwS only swaps the mapping of two sets at once, and all cache sets are automatically shifted after multiple swaps.

A global counter is used in SwS to store the number of writes to the cache, which we annotate as Wr. Two registers, SwV (changing from 0 to N-2) and ShV (changing from 0 to N-1), are used to track the current status of the swaps and shifts, respectively. The detailed mechanism is explained as below:

Swap Round (SwapR): Every time Wr reaches a specific threshold (Swap Threshold, ST), a swap between cache set [SwS] and set [SwS+1] is triggered. Note that this swap operation only exchanges the set IDs, and the data stored in these two sets are simply invalidated (or written-back to the next level of caches if dirty). After that SwS is incremented by 1. One swap round (SwapR) consists of N-1 swaps. One SwapR actually indicates that all the cache set IDs are shifted by 1.

Shift Round (ShiftR): ShV is incremented by 1 after each SwapR. At the same time, SwS is reset to 0. One shift round (ShiftR) consists of N shifts (i.e. SwapR).

Figure 5 shows an example of the set ID mapping under different SwV and ShV for a cache consisting of 4 sets. It shows the process of one complete ShiftR, which consists of 4 SwapR. It can be seen that one SwapR consists of 3 swaps and all cache sets are shifted by 1 after each SwapR. In addition, after one ShiftR, all cache sets are shifted to the original position and all logical set indexes are the same as the physical ones.

The performance penalty of SwS is small because only two sets are swapped at once and the swap interval period can be long enough (e.g. million cycles) by adjusting ST. The performance analysis of SwS is in Section 9.1.

SwS Implementation: The implementation of SwS is shown in Figure 6. One global counter is used to store wr and two registers are used to store ShV and SwV. When a logical set number (LS) comes, the physical set number (PS) can be computed based on three different situations:

1. If $LS = SwV$, it means this logical set is exactly the

cache set should be swapped in this ShiftR. Therefore, PS is mapped to the current shift value (ShV).

2. If $LS > SwV$, it means this set has not been shifted in this ShiftR. Therefore, PS is mapped to $LS + ShV$.
3. If $LS < SwV$, it means this set has been already shifted. Therefore, PS is mapped to $LS + ShV + 1$.

When a dirty cache line is written back to the next level of cache, the logical set address needs to be re-generated. The mapping from PS to LS is symmetrical and is also given in Figure 6. This mapping policy can be verified by the simple example in Figure 5. Because SwV and ShV are changed along with cache writes, the mapping between LS and PS change all the time. This ensures that the writes to different physical sets are balanced, reducing inter-set variations.

Compared to a conventional cache architecture, the set index translation step in SwS only adds a simple arithmetic operation and can be merged into the row-decoder. We implemented the LS-to-PS address translation in 45nm CMOS technology and it can be completed well in one cycle with 3GHz clock frequency. In addition, this 1-cycle latency overhead is only paid on higher-level cache misses which need to access lower-level caches.

6 Intra-Set Variation: A More Severe Issue

SwS can distribute writes to all the cache sets, but it only reduces the inter-set write variation. Our experimental results later in Section 8.2 show that using SwS cannot eliminate intra-set variations. Thus, we need other schemes using principles fundamentally different from previous wear-leveling techniques to reduce intra-set variations.

6.1 Set Line Flush

Intra-set write variations are mainly caused by hot data being written more frequently than others. For example, if one cache line in a set is frequently targeted by cache write hits and absorbs a large number of cache writes, write accesses may be unevenly distributed in this cache set.

In a traditional LRU (least recently used) cache policy, every accessed block is marked towards MRU (most recently used) to avoid being chosen as the victim line. As a result, the LRU policy rarely replaces the hot data since it is frequently accessed by cache write hits. This increases the write count of one block and the intra-set write variation of the corresponding set.

To solve this problem, we first consider a *set line flush* (LF) scheme. In LF, when a cache write hit happens, the new data is put into the write-back buffer directly instead of writing it to the hit data block, and then the cache line is marked as *INVALID*. This process is called *set line flush*. Hence, the block containing the hot data has the opportunity to be replaced by other cold data according to the LRU policy, and the hot data can be reloaded to other cache lines. We invalidate the hot data line instead of moving it to other positions due to similar reasons given in Section 5.1 showing that data invalidation is better than data movement.

LF balances the write count to each cache block, but it flushes every cache write hit no matter whether it contains hot data or not. Obviously, LF causes large performance degradation as the flushed data has to be reloaded if it is hot. To reduce the performance penalty, we need to add intra-set write count statistics and only flush hot cache lines.

6.2 Hot Set Line Flush

One of the simplest solutions is to add counters in the cache to store the write count of every cache line, and we call this scheme *hot set line flush* (HoLF). Theoretically, if the difference of the largest write count of one line and the average value of the set is beyond a predetermined threshold, it means that the data in this cache line is too hot and should be flushed.

However, HoLF has significant overhead in both area and performance:

- The area overhead is large since it requires adding one counter for every cache line. Considering the cache line is 64B wide and the write counter is 20-bit, the hardware overhead is more than 3.7%.
- The performance is inevitably impaired because HoLF tracks both maximum and average write count values in every cache set. It is infeasible to initiate multiple arithmetic calculations for every cache write.

We do not discuss HoLF further in this paper as it is not a practical solution. Instead, we introduce an improved solution called *probabilistic set line flush* (PoLF).

6.3 Probabilistic Set Line Flush

The motivation of *probabilistic set line flush* (PoLF) is to flush hot data probabilistically instead of deterministically.

Probabilistic Invalidation: Unlike HoLF, PoLF only maintains one global counter to count the number of write hits to the entire cache, and it flushes a cache line when the counter saturates no matter whether the cache line to be flushed is hot or not. Although there is no guarantee that the hottest data would be flushed as we desire, the probability of PoLF selecting a hot data line is high: the hotter the data is, the more likely it will be selected when the global counter saturates. Theoretically, PoLF is able to flush the hottest data in a cache set most of the time, and the big advantage of PoLF is that it only requires one global counter.

Maintaining LRU: For normal LRU policy, when a cache line is invalidated, the age bits of this line is marked as LRU. However, for PoLF, because hot data are accessed

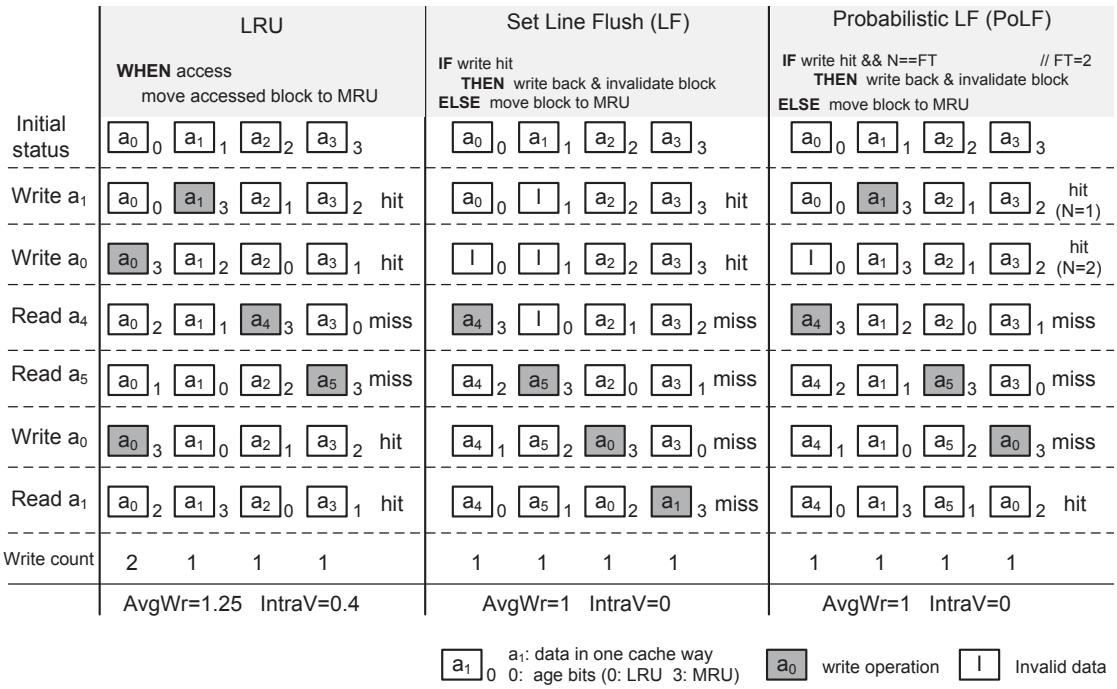

Figure 7. The behavior of one cache set composed of 4 ways under LRU, LF and PoLF polices for the same access pattern. The total write count of each cache way, the average write count and the intra-set variation are marked, respectively.

more frequently, it is possible that after invalidating a single hot cache line, the same data will be reinstalled in the very same line on a subsequent miss. Thus, we modify PoLF to maintain age bits for probabilistic invalidations. When PoLF flushes a line, it does not mark it as the LRU line and its age bits are not changed. Later, a subsequent miss will invalidate the actual LRU line and reinstall the hot data in that line. The cache line evicted by PoLF remains invalid until it becomes the LRU line.

Comparison with Other Policies: Figure 7 shows the behavior of a 4-way cache set under LRU, LF, and PoLF polices for an exemplary access pattern. The following observations can be seen from this example:

- For LRU, the hot data a_0 is moved to the MRU position (age bits=3) after each write hit and is never replaced by other data. Thus, the intra-set variation using LRU is the largest one among all the polices.
- For LF, each write hit causes its corresponding cache line to be flushed. The age bits are not changed during write hits. The intra-set variation is reduced compared to the LRU policy because the hot data a_0 is reloaded into another cache line. However, data a_1 is also flushed since every write hit causes one cache line flush, and it brings one additional access miss.
- For PoLF, we let every other write hit cause a cache line flush (i.e. line-flush threshold FT=2)[4]. Compared to the LRU policy, its intra-set variation is reduced because the hot data a_0 is moved to another cache line. In addition, compared to LF, the number of misses is reduced because a_1 is not flushed.

[4] FT is set as 2 for this illustration. More typical values are much larger and shown in Section 8.

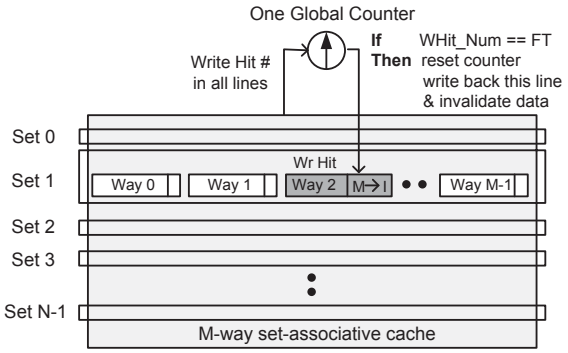

Figure 8. The cache architecture of PoLF. Only one global write hit counter is added to the entire cache.

Thus, PoLF can reduce the intra-set variation as well as ensuring the probability of selecting a hot data line is high.

PoLF Implementation: We can design the hardware implementation for PoLF as follows. First, we add a write hit counter (one counter for the entire cache). The counter is only incremented at each write hit event. If the counter saturates at one threshold, then the cache will record the write operation that causes the counter saturation, and invalidate the corresponding line to the write hit. Figure 8 shows the architecture of PoLF. The only hardware overhead of PoLF is a global counter that tracks the total

number of write hits to the cache. The tunable parameter of PoLF is the line-flush threshold FT.

7 i^{2}WAP: Putting Them Together

We combine SwS and PoLF together to form our inter- and intra-set write variation-aware policy, i^{2}WAP. In i^{2}WAP, SwS and PoLF work independently: SwS reduces inter-set variations, and PoLF reduces intra-set variations. The total write variations can be reduced significantly and the product lifetime can be improved. Moreover, the implementation overhead of i^{2}WAP is quite small: SwS only requires one global counter to store the number of write accesses and two registers to store the current swapping and shifting values; PoLF only needs another global counter to store the number of write hit accesses.

8 Experiments

In this section, we first describe our experiment methodology, then we demonstrate how SwS and PoLF reduce the inter- and intra-set write variations, respectively. Finally, we show how i^{2}WAP improves the non-volatile cache lifetime.

8.1 Baseline Configuration

Our baseline is a 4-core CMP system. Each core consists of private L1 and L2 caches and all the cores share an L3 cache. Our experiment makes use of a 4-thread OpenMP version of the PARSEC 2.1 [2] benchmark workloads. We run single application since wear leveling techniques are usually designed for the worst case. The native inputs are used for the PARSEC benchmark to generate realistic program behavior. We modify the gem5 full-system simulator [3] to implement our proposed techniques and use it to collect cache accesses. Each gem5 simulation run is fast forwarded to the pre-defined breakpoint at the code region of interest, warmed-up by 100 million instructions, and then simulated for at least 10 billion instructions. The characteristics of workloads are listed in Table 1, in which WPKI and TPKI are writes and transactions per kilo-instructions, respectively.

In this work, we use ReRAM L2 and L3 caches as an example. Our techniques and evaluations are also applicable to other non-volatile memory technologies. The system parameters are given in Table 2.

8.2 Write Variations Reduction

8.2.1 Effect of SwS on Inter-Set Variations.

In SwS, the inter-set variation reduction is related to the number of shift rounds (ShiftR) during the experimental time. Assuming there are N sets in the cache, one ShiftR includes N swap rounds (SwapR) and one SwapR has $N-1$ swaps. After each ShiftR, all the cache sets are shifted through all the possible locations. Thus, the more rounds the cache is shifted, the more evenly the write accesses are distributed to each cache set.

Table 1. Workload characteristics in L2 an L3 caches under our baseline configuration.

	L2 cache		L3 cache	
Workload	WPKI	TPKI	WPKI	TPKI
blackscholes	0.07	0.4	0.04	0.3
canneal	0.04	23	0.01	15
dedup	1.1	4.8	0.4	0.8
facesim	3.3	4.7	1.1	1.4
ferret	1.8	6.3	0.2	0.5
fluidanimate	0.4	1.4	0.3	0.8
freqmine	1.3	6.7	0.2	0.4
raytrace	0.56	0.62	0.03	0.25
streamcluster	3.7	4.2	0.9	1.1
swaptions	1.4	2.9	0.02	0.06
vips	1.1	4.4	0.6	1.0
x264	0.7	16.1	0.2	0.5

We annotate the round number of ShiftR as RRN, and it can be computed as follows:

$$RRN = \frac{W_{total}}{ST \times N \times (N-1)} = \frac{WPI \times I_n}{ST \times N \times (N-1)} \quad (6)$$

in which ST is the swap threshold, W_{total} is the product of WPI (write access per instruction) and I_n (the number of simulation instructions). For the same application, if the execution time is longer, which means I_n is larger, we can use a larger ST value to get the same RRN.

To illustrate the relationship between the inter-set variation reduction and RRN, we run simulations using different configurations with different execution lengths. Figure 9 shows the result. We can see when RRN is increased, the inter-set variation is reduced significantly. When RRN is larger than 100, the inter-set variation can be reduced to smaller than 5% of its original value.

For a 1MB cache running an application with WPKI equal to 1, if we want to reduce its inter-set variation by 95% within 1 month, then the swap threshold ST can be set larger than 100,000 according to Eqn. 6 by setting RRN as 100. However, simulating a system within 1-month wall clock time is never realistic. To evaluate the effectiveness of SwS, we use a smaller ST (e.g. ST=10) in a relatively shorter period of execution time (e.g. 100 billion instructions) to get a similar RRN. Figure 9 (b) shows the inter-set variation of an L2 cache after adopting SwS when RRN equals to 100. The average inter-set variation is significantly reduced from 66% to 1.2%.

In practice, ST can be scaled along with the entire product lifespan since our wear-leveling goal is to balance the cache line write count in the scale of several months if not years. Thus, the swap operation in SwS is infrequent enough to hide its impact on system performance.

Table 2. Baseline configurations.

System	4-core, 3GHz, out-of-order CPU model based on ALPHA 21264
SRAM* I-L1/D-L1 caches	private, 32KB/32KB, 8-way, 64-Byte cache line, LRU & write-back, write allocate, 2-cycle
ReRAM L2 cache	private, 1MB, 8-way, 64-Byte cache line, LRU & write-back, write allocate, 30-cycle
ReRAM L3 cache	shared, 8MB, 8-way, 64-Byte cache line, LRU & write-back, write allocate, 100-cycle
DRAM main memory	4GB, 128-entry write buffer, 200-cycle

* We envision SRAM is still used in L1 due to the concerns of performance and the current ReRAM write endurance limit.

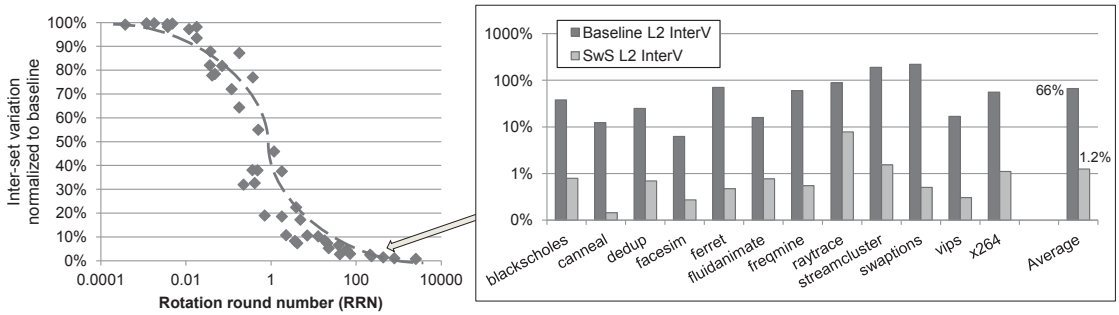

Figure 9. Inter-set variations normalized to baseline when RRN increases in SwS scheme. The zoom-in sub-figure shows the detailed L2 inter-set variation of different workloads after adopting the SwS scheme when RRN equals to 100.

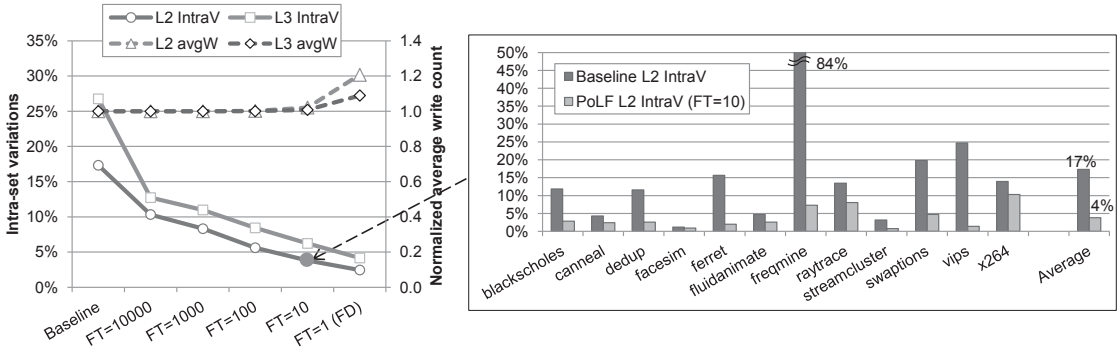

Figure 10. The average intra-set variation and the average write count normalized to baseline for L2 and L3 caches after adopting a PoLF scheme. The zoom-in sub-figure shows the detailed L2 intra-set variation for different workloads after adopting an PoLF scheme with line-flush threshold (FT) of 10.

8.2.2 Effect of PoLF on Intra-Set Variations.

Figure 10 shows how PoLF affects intra-set variations and average write counts for L2 and L3 caches. It can be seen that PoLF reduces the intra-set variation significantly and the strength of PoLF is changed with different FT values.

When FT equals to 1, the PoLF scheme flushes every write hit and it is equivalent to the LF scheme. Figure 10 shows that LF can further reduce intra-set variations compared to PoLF. However, the average write count of LF is increased significantly. Thus, considering the impact on both intra-set variations and average write counts, we choose PoLF with FT that equals to 10. The results show that by adopting PoLF, the average intra-set variation of L2 cache can be reduced from 17% to 4%, and the average intra-set variation of L3 cache can be reduced from 27% to 6% with a FT value of 10. The average write count is increased by less than 2% compared to the baseline.

8.2.3 Effect of i^{2}WAP on Total Variations.

Figure 11 shows the total variations of L2 and L3 caches under different policies. Compared to the baseline, SwS reduces inter-set variations across all workloads, but it does not affect intra-set variations. On the other hand, PoLF reduces intra-set variations with small impact on inter-set variations. By combining SwS and PoLF, i^2WAP reduces the total variations significantly. Figure 11 shows that on average the total variation is reduced from 84% to 5% for L2 caches and from 49% to 15% for L3 caches.

8.3 Lifetime Improvement of i^{2}WAP

After reducing inter-set and intra-set variations, the lifetime of non-volatile caches can be improved. Figure 12 shows the lifetime improvement of L2 and L3 caches after adopting SwS only, PoLF only, and the combined i^{2}WAP policy, respectively. The lifetime improvement varies based

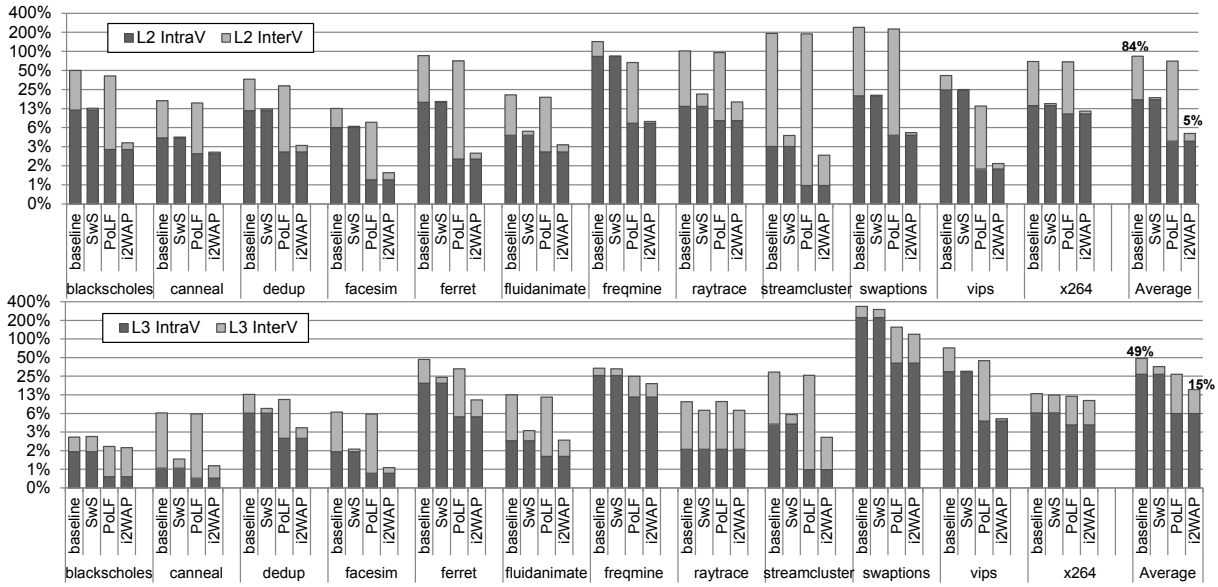

Figure 11. The total variation for L2 and L3 caches under the baseline configuration, SwS scheme (RRN=100), PoLF scheme (FT=10) and i^{2}WAP policy. Each value is broken down to the inter-set variation and the intra-set variation. Note that a log scale is used to cover a large range of variations.

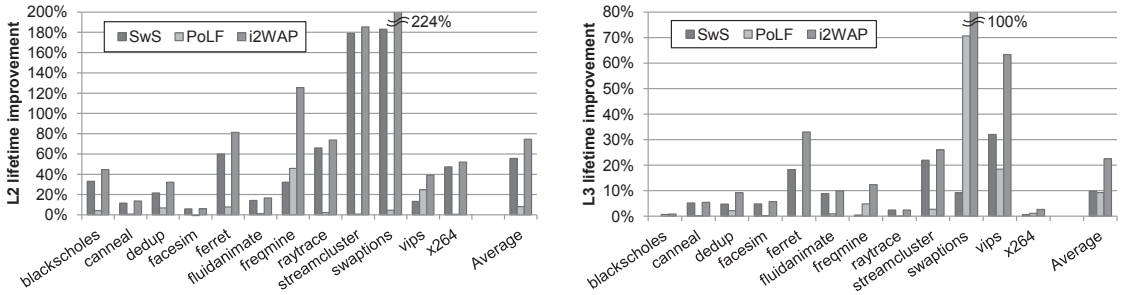

Figure 12. The lifetime improvement after adopting i^{2}WAP using Eqn. 5. (Left: L2, Right: L3)

on the workload. Basically, the larger the original variation value is, the bigger the improvement a workload has. The overall lifetime improvement is 75% (up to 224%) for L2 caches and 23% (up to 100%) for L3 caches.

9 Analysis of Other Issues

9.1 Performance Overhead of i^{2}WAP

Since i^{2}WAP causes extra cache invalidations, it is necessary to compare its performance to a baseline system without wear-leveling[5]. Figure 13 shows the performance overhead of a system in which L2 and L3 caches using i^{2}WAP with $ST = 100,000$ and $FT = 10$ compared to a baseline system in which an LRU policy is adopted.

As shown in Figure 13, on average, the IPC of the system using i^{2}WAP is reduced only by 0.25% compared to the baseline. The performance penalty of i^{2}WAP is very small because of two reasons:

- In SwS, the interval of swap operations is long (e.g. 10 million instructions), and only two cache sets are re-mapped for each operation.
- In PoLF, write hit accesses are infrequent enough to ensure the frequency of set line flush operations is low (e.g. once per 10^5 instructions), and only one cache line is flushed each time. In addition, designers can tradeoff between the number of flush operations and the variation value by adjusting FT.

9.2 Impact on Main Memory

We also evaluate the impact of extra write backs on main memory. Figure 14 shows the write count to the main memory compared to the baseline system after adopting the i^{2}WAP policy. The result shows that its impact on the write count is very small, only increasing about 4.5% on average. For most workloads, the write count is increased by less than 1%. In addition, because most writes can be filtered by caches and the write count of main memory is much smaller

[5]The simulator has a protocol to ensure cache coherency when invalidations occur, thus the performance overhead of this part is included.

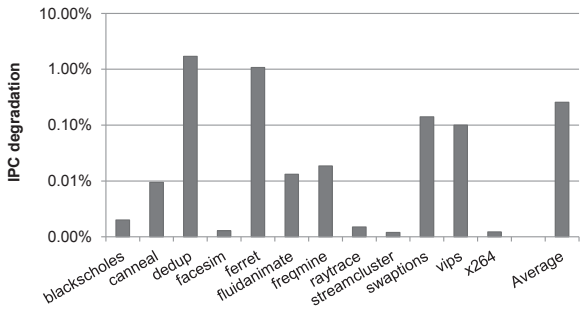

Figure 13. The system IPC degradation compared to the baseline system after adopting the i^{2}WAP policy.

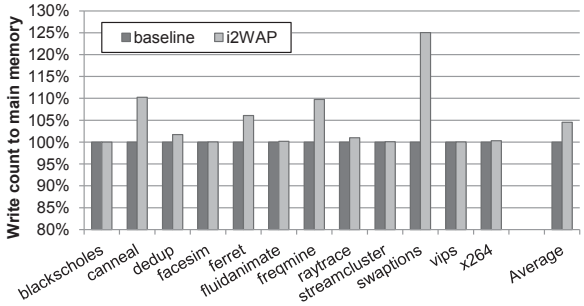

Figure 14. The write count to the main memory normalized to the baseline system after adopting i^{2}WAP.

than that of caches, the endurance requirement for non-volatile main memory is much looser. In the worst case, although the write count to the main memory of $swaptions$ is increased by 25%, its absolute value of write backs is very small (about 0.001 writes to the main memory per kilo-instructions). Thus, it does not significantly degrade the lifetime of the main memory even though its write access frequency is relatively higher.

9.3 Error-Tolerant Lifetime

While our analyses are all focused on the raw cache lifetime, this lifetime can be easily extended by tolerating partial cell failures. There are two factors causing the different failure time of cells. The first one is the variation of write counts, which is addressed mainly in this work. The second one is the inherent variation of the cell's lifetime due to process variations, which needs another type of techniques to solve (discussed in Section 10). For both factors, the system lifetime can be extended by tolerating a small number of cell failures.

It is much simpler to extend i^{2}WAP and tolerate the failed cache lines comparing to tolerating main memory failures [8, 17, 19, 20, 26]. We can force the failed cache lines to be tagged *INVALID*, so that no further data would be written to the failed cache lines. In this case, the number of ways in the corresponding cache set is only reduced by

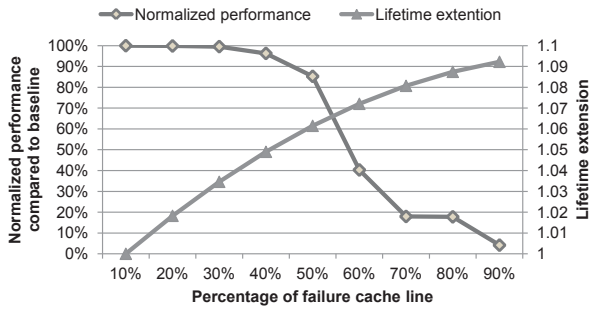

Figure 15. The performance degradation and lifetime extension during gradual cache line failure on a non-volatile cache hierarchy.

1, e.g. from 8-way associative to 7-way associative.

The error-tolerant lifetime is at least the same as the raw lifetime and may be much longer. However, the performance is degraded because the cache associativity is reduced. Figure 15 shows an analysis of an ReRAM-based cache hierarchy with 32KB L1 caches, 1MB L2 caches, and 8MB L3 caches. It shows that if the system can tolerate the failure of 50% of the cache lines at all levels, the lifetime can be extended by 6% and the performance penalty is 15%[6].

10 Related Work

There are various previous architectural proposals to extend the lifetime of non-volatile memories. These prior work can be classified by two basic types of techniques:

The first category focuses on evenly distributing unbalanced write frequencies to all memory lines. Zhou *et al.* [27] proposed a segment swapping policy for PCM main memory. Qureshi *et al.* proposed fine-grain wear leveling (FGWL) [16] and start-gap wear leveling [15] to shift cache lines within a page to achieve uniform wear out of all lines in the page. Seong *et al.* [20] addressed potential attacks by applying security refresh. However, this previous work was all focused on extending the lifetime of PCM-based main memory. Other work on non-volatile caches [9] only extended wear-leveling techniques for main memory without considering the different operating mechanisms of main memory and caches. As discussed in Section 5.2, these techniques only reduce the inter-set variation when they are adopted in caches. The cache intra-set variation is a new problem and is not considered in previous work.

The second category is about error corrections. The conventional Error Correction Code (ECC) is the most common technique in this category. Dynamically replicated memory [8] reuses memory pages containing hard faults by

[6]The value of performance degradation and the lifetime extension depend on the cache hierarchy and capacity, but the trends for different configurations are similar.

dynamically forming pairs of pages that act as a single one. Error Correction Pointer (ECP) [17] corrects failed bits in a memory line by recording the position and its correct value. Seong et al. [19] propose SAFER to efficiently partition a faulty line into groups and correct the error in the group. FREE-p [26] proposed an efficient means of implementing line sparing. These architectural techniques add different types of data redundancy to solve the access errors caused by limited write endurance.

It should be noted that all the techniques from the second category are orthogonal to our proposed policies, and the total system lifetime could be extended further after combining i^{2}WAP with them.

11 Conclusion

Non-volatile memories are promising technologies for replacing SRAM or eDRAM in low-level on-chip caches due to their advantages in reducing leakage or refresh power as well as their higher density. However, non-volatile memory technologies usually have limited write endurance. Moreover, the existing wear-leveling techniques for non-volatile main memories cannot be effectively applied to on-chip non-volatile caches because caches have intra-set write variations in addition to inter-set variations. In this work, we propose i^{2}WAP, a new endurance-aware cache management policy. i^{2}WAP uses SwS to reduce the inter-set variation and PoLF to reduce the intra-set variation. i^{2}WAP can balance the write traffic to each cache line, greatly reducing both intra-set and inter-set variations, and thus improving the lifetime of non-volatile on-chip L2 and L3 caches. The implementation overhead of i^{2}WAP is small, only needing two global counters and two global registers, but it can improve the lifetime of non-volatile caches by 75% on average and up to 224% over the conventional cache management policies.

12 Acknowledgements

This work is supported in part by SRC grants, NSF 1218867, 1213052, 0903432 and by DoE under Award Number DE-SC0005026.

References

[1] A. Akel et al. Onyx: A protoype phase-change memory storage array. In *FAST*, 2011.

[2] C. Bienia et al. The PARSEC benchmark suite: characterization and architectural implications. In *PACT*, 2008.

[3] N. Binkert et al. gem5:A Multiple-ISA Full System Simulator with Detailed Memory Model. In *Computer Architecture News*, 2011.

[4] A. M. Caulfield et al. Moneta: A high-performance storage array architecture for next-generation, non-volatile memories. In *MICRO*, 2010.

[5] J. Condit et al. Better I/O through byte-addressable, persistent memory. In *SOSP*, 2009.

[6] Y. Huai. Spin-transfer torque MRAM (STT-MRAM): Challenges and prospects. *AAPPS Bulletin*, 18(6), 2008.

[7] International Technology Roadmap for Semiconductors. Process Integration, Devices, and Structures 2010 Update. `http://www.itrs.net/`.

[8] E. Ipek et al. Dynamically replicated memory: Building reliable systems from nanoscale resistive memories. In *ASPLOS*, 2010.

[9] Y. Joo et al. Energy- and endurance-aware design of phase change memory caches. In *DATE*, 2010.

[10] C. H. Kim et al. A forward body-biased low-leakage SRAM cache: device and architecture considerations. In *ISLPED*, 2003.

[11] Y.-B. Kim et al. Bi-layered rram with unlimited endurance and extremely uniform switching. In *VLSI*, 2011.

[12] D. Kroft. Lockup-free instruction fetch/prefetch cache organization. In *ISCA*, 1998.

[13] B. C. Lee et al. Architecting phase change memory as a scalable DRAM alternative. In *ISCA*, 2009.

[14] W. S. Lin et al. Evidence and solution of over-RESET problem for HfO_x based resistive memory with sub-ns switching speed and high endurance. In *IEDM*, 2010.

[15] M. K. Qureshi et al. Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling. In *MICRO*, 2009.

[16] M. K. Qureshi et al. Scalable high performance main memory system using phase-change memory technology. In *ISCA*, 2009.

[17] S. Schechter et al. Use ECP, not ECC, for hard failures in resistive memories. In *ISCA*, 2010.

[18] S. Segars. Low power design techniques for microprocessors. In *ISSCC*, 2001.

[19] N. H. Seong et al. SAFER: Stuck-at-fault error recovery for memories. In *MICRO*, 2010.

[20] N. H. Seong et al. Security refresh: prevent malicious wear-out and increase durability for phase-change memory with dynamically randomized address mapping. In *ISCA*, 2010.

[21] S.-S. Sheu et al. A 4Mb embedded SLC resistive-RAM macro with 7.2ns read-write random-access time and 160ns MLC-access capability. In *ISSCC*, 2011.

[22] Y.-H. Tseng et al. High density and ultra small cell size of contact ReRAM (CR-RAM) in 90nm CMOS logic technology and circuits. In *IEDM*, 2009.

[23] R. Waser and M. Aono. Nanoionics-based resistive switching memories. *Nature Materials*, 6(11), 2007.

[24] J. J. Yang et al. Memristive switching mechanism for metal/oxide/metal nanodevices. *Nature Nanotechnology*, 3(7), 2008.

[25] J. J. Yang et al. High switching endurance in TaO_x memristive devices. *Applied Physics Letters*, 97(23), 2010.

[26] D. H. Yoon et al. FREE-p: Protecting non-volatile memory against both hard and soft errors. In *HPCA*, 2011.

[27] P. Zhou et al. A durable and energy efficient main memory using phase change memory technology. In *ISCA*, 2009.