Chapter 9
Wear-Leveling Techniques for Nonvolatile Memories

Jue Wang, Xiangyu Dong, Yuan Xie and Norman P. Jouppi

Abstract Nonvolatile memories (NVMs) are promising technologies for replacing SRAM or eDRAM in low-level on-chip caches and main memories because they can save standby power and provide high cache capacity. However, limited write endurance is a common problem for NVM technologies. The current memory management policies are not write variation aware and result in significant nonuniformity in terms of writing to memory blocks, which would cause heavily written nonvolatile cache blocks to fail much earlier than most other blocks. Thus, wear-leveling techniques are important for NVM-based memory system to balance write traffic and extend the system lifetime. Some wear-leveling techniques have been proposed for NVM-based main memories and on-chip caches. In this chapter, we use inter-/intra set write variation aware cache policy (i^2WAP) as an example to show how to design an endurance-aware management policy for nonvolatile caches. i^2WAP has two features: first, Swap-Shift (SwS), an enhancement based on previous main memory wear leveling to reduce cache inter-set write variations; second, probabilistic set line flush (PoLF), a novel technique to reduce cache intra-set write variations. Implementing i^2WAP only needs two global counters and two global registers. By adopting i^2WAP, we can improve the lifetime of on-chip nonvolatile caches by 75 % on average and up to 224 %.

J. Wang · Y. Xie (✉)
Pennsylvania State University, Pennsylvania, US
e-mail: yuanxie@cse.psu.edu

J. Wang
e-mail: jzw175@cse.psu.edu

X. Dong
Qualcomm Technology, Inc, San Diego, US
e-mail: xiangyud@qualcomm.com

N. P. Jouppi
Google, Inc., CA, USA
e-mail: jouppi@acm.org

Y. Xie (ed.), *Emerging Memory Technologies*,
DOI: 10.1007/978-1-4419-9551-3_9,

9.1 Overview of Wear-Leveling Techniques

The scaling of SRAM and DRAM is increasingly constrained by technology limitations such as leakage power and cell density. Recently, some emerging nonvolatile memory (NVM) 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 nonvolatile memory technologies have advantages in high density, low standby power, better scalability, and nonvolatility.

However, adoption of NVM at the main memory or on-chip cache level has a write endurance issue. For example, PCM is only expected to sustain 10^8 writes before experiencing frequent stuck-at-1 or stuck-at-0 errors [6, 12, 14–16, 21]. For ReRAM, the write endurance bar is much improved but is still around 10^{11} [9]. 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 [4]. Although the absolute write endurance values for NVMs seem sufficiently high for use, the actual problem is that the current cache management policies are not write variation aware. These polices were originally designed for SRAM/DRAM and result in significant nonuniformity in terms of writing to memory blocks, which would cause heavily written NVM blocks to fail much earlier than most other blocks.

There are various previous architectural proposals to extend NVM lifetimes. These work can be classified by two basic types of techniques: The first category is wear-leveling technique which focuses on evenly distributing unbalanced write frequencies to all memory lines. Zhou et al. [22] proposed a row shifting and a segment swapping policy for PCM main memory. Their key idea is to periodically shift the memory row and swap the memory segments to even out the write accesses. Qureshi et al. [13] proposed fine-grain wear leveling (FGWL) and start-gap wear leveling [12] to shift cache lines within a page to achieve uniform wear out of all lines in the page. Seong et al. [15] addressed potential attacks by applying security refresh. They used a dynamic randomized address mapping scheme that swaps data using random keys upon each refresh due. There are also some other work for NVM caches. Joo et al. [7] extended wear-leveling techniques for main memory and proposed the bit-line shifting and word-line remapping techniques for nonvolatile caches. Wang et al. [20] proposed i^2WAP to minimize both inter- and intra-set write variations, and in this chapter we will give detail description of the method proposed in [20]

The second category is about error corrections. The conventional error correction code (ECC) is the most common technique in this category. Dynamically replicated memory [6] reuses memory pages containing hard faults by dynamically forming pairs of pages that act as a single one. Error correction pointer (ECP) [14] corrects failed bits in a memory line by recording the position and its correct value. Seong et al. [16] propose SAFER to efficiently partition a faulty line into groups and correct the error in the group. FREE-p [21] 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 the wear-leveling techniques from the first category. And the total system lifetime could be extended further after combining them together.

9.2 Background

Write endurance is defined as the number of times a memory cell can be overwritten, and emerging NVMs commonly have a limited write endurance. In this section, the failure mechanisms of different NVMs are introduced.

9.2.1 PCM Failure Mechanism

PCM devices have two major failure modes: *stuck-RESET* and *stuck-SET* [8]. Stuck-RESET failures are caused by void formation or delamination that catastrophically disconnects the electrical path between GST (i.e., the storage element of PCM) and access device. Instead, stuck-SET failures are caused by the aging of GST material that becomes more reluctant to create an amorphous (RESET) phase after continuously experiencing write cycles, resulting in a degradation of the PCM RESET-to-SET resistance ratio. Both of these failure modes are commonly observed in PCM devices. ITRS [5] projects that the average PCM write endurance is in a range between 10^7 and 10^8.

9.2.2 ReRAM Failure Mechanism

There are several different types of endurance failure mechanisms are observed in ReRAM devices. One of the them is anode oxidation-induced interface reaction [3]. High temperature, large current/power process and oxygen ions produced during forming/SET process cause the oxidation at the anode–electrode interface. Another endurance failure mechanism is extra vacancy attributed reset failure effect [3]. Electric field-induced extra oxygen vacancy generation during switching may increase the filament size or make the filament rougher, accompanying with the reduced resistance in high-resistance states (R_{HRS}) and resistance in low-resistance states (R_{LRS}) as well as the increased reset voltage. Recent ReRAM prototypes demonstrate the best write endurance ranging from 10^{10} [11] to 10^{11} [9].

9.2.3 STTRAM Failure Mechanism

STTRAM uses magnetic tunnel junction (MTJ) as the storage element, and there are two challenges in controlling the MTJ reliability: time-dependent dielectric breakdown (TDDB) and resistance drift [18]. TDDB is detected as an abrupt increase in junction current owing to a short forming through the tunneling barrier. Resistance drift is a gradual reduction in the junction resistance over time that can eventually lead to reduced read margin. The best endurance test of STTRAM so far is less than 4×10^{12} [4].

9.3 Improving Nonvolatile Cache Lifetime by Reducing Inter- and Intra-set Write Variations

The write variation brought by normal memory management policy would cause heavily written memory blocks to fail much earlier than their expected lifetime. Thus, eliminating memory write variation is one of the most critical problems that must be addressed before NVMs can be used to build practical and reliable memory systems.

There are many wear-leveling techniques to extend the lifetime of nonvolatile main memories and caches. In this chapter, we use inter-/intra-set Write variation-aware cache policy (i^2WAP) [20] as an example to show how to improve nonvolatile cache lifetime by reducing write variations.

The difference between cache and main memory operational mechanisms makes the wear-leveling techniques for NVM main memories inadequate for NVM 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 NVM caches. In order to demonstrate how severe the problem is for NVM caches, we first do a quick experiment.

9.3.1 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:

$$\text{InterV} = \frac{1}{W_{\text{aver}}} \sqrt{\frac{\sum_{i=1}^{N} \left(\sum_{j=1}^{M} w_{i,j}/M - W_{\text{aver}} \right)^2}{N-1}} \tag{9.1}$$

$$\text{IntraV} = \frac{1}{W_{\text{aver}} \times N} \sum_{i=1}^{N} \sqrt{\frac{\sum_{j=1}^{M} \left(w_{i,j} - \sum_{j=1}^{M} w_{i,j}/M \right)^2}{M-1}} \tag{9.2}$$

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

$$W_{\text{aver}} = \frac{\sum_{i=1}^{N} \sum_{j=1}^{M} w_{i,j}}{NM} \tag{9.3}$$

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

Figure 9.1 shows the experimental results of InterV and IntraV in our simulated 4-core system with 32 KB I-L1, 32 KB D-L1, 1 MB L2, and 8 MB L3 caches. The detailed simulation methodology and the setting are described in Sect. 9.8. We compare InterV and IntraV in L2 and L3 caches as we assume that emerging NVM will be first used in low-level caches. From Fig. 9.1, we make two observations:

(1) There are 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.

(2) There are 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

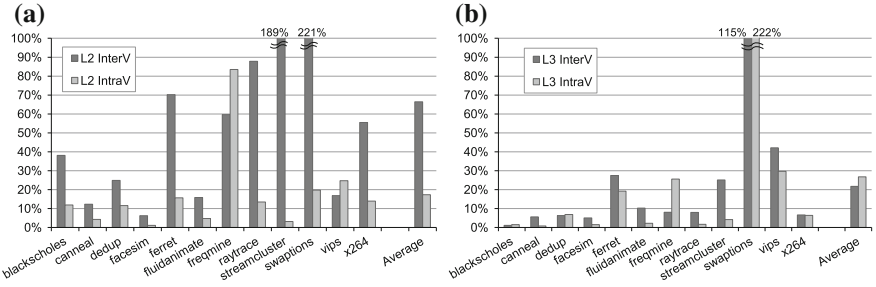

Fig. 9.1 The coefficient of variation for inter-set and intra-set write count of L2 and L3 caches in a simulated 4-core system with 32 KB I-L1, 32 KB D-L1, 1 MB L2, and 8 MB L3 caches

[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.

thus the write accesses may be unevenly distributed to the remaining M-1 lines in the set (for an M-way associative cache), e.g., *freqmine* has 84 % IntraV in L2, and *swaptions* 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, L3 caches have smaller InterV than L2 caches do. This is because L2 caches are private for each processor but all processors share one L3 cache, and L3 has less write variance 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 that IntraV is roughly the same or even larger compared to InterV for some workloads. Combining these two types of write variations together significantly shorten the NVM cache lifetime.

9.4 Cache Lifetime Metrics

Cache lifetime can be defined in two different ways: *raw lifetime* and *error-tolerant lifetime*. We define the raw lifetime by the first failure of a cache line without considering any error recovery effort. On the other hand, we can extend the raw lifetime by using error correction techniques and paying overhead in either memory performance or memory capacity [6, 14, 16, 21], and we call it the error-tolerant lifetime. In this work, we focus on how to improve the raw lifetime at first as it is the base of the error-tolerant lifetime. Later, we discuss the error-tolerant lifetime in Sect. 9.10.3.

The target of maximizing the cache raw lifetime is equivalent to minimizing the worst-case write count to a cache line. However, it is impractical to obtain the worst-case write count throughout the whole product lifetime which might span several years. Instead, in this work, we model the raw lifetime by using three parameters: *average write count*, *inter-set write count variation*, and *intra-set write count variation*. We first sample the statistics of the cache write access behavior during a short period time of simulation, calculate the statistical estimation of these three parameters, and then predict the cache raw lifetime based on statistical estimations. Our methodology can be explained in details as follows:

(1) The cache behavior is simulated during a short period of time t_{sim} (e.g., 10 billion instructions on a 3 GHz CPU).
(2) Each cache line write count is collected to get a average write count W_{aver}. Also, we calculate InterV and IntraV according to Eqs. 9.1 and 9.2.

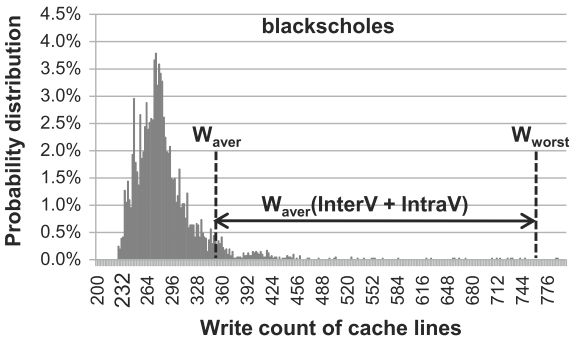

Fig. 9.2 The L2 cache write count probability distribution function (PDF) of blackscholes

(3) Assuming the total write variation of a cache line is the summation of its inter- and intra-set variations,[2] we then have $W_{\text{var}} = W_{\text{aver}} \times (\text{InterV} + \text{IntraV})$.
(4) The worst-case write count is predicted as $W_{\text{aver}} + W_{\text{var}}$ to make sure to cover the vast majority of cases. While this approach is approximate, Fig. 9.2 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

$$t_{\text{total}} = \frac{W_{\max} \times t_{\text{sim}}}{W_{\text{aver}} + W_{\text{var}}} = \frac{W_{\max} \times t_{\text{sim}}}{W_{\text{aver}}(1 + \text{InterV} + \text{IntraV})} \tag{9.4}$$

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

$$\text{LI} = \frac{W_{\text{aver\_base}}(1 + \text{InterV}_{\text{base}} + \text{IntraV}_{\text{base}})}{W_{\text{aver\_imp}}(1 + \text{InterV}_{\text{imp}} + \text{IntraV}_{\text{imp}})} - 1 \tag{9.5}$$

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.

We apply the state-of-the-art least recently used (LRU) cache management policy as the baseline, and Fig. 9.3 shows how bad the baseline is in terms of write variation and cache raw lifetime. Compared to the ideal case where all the write traffic is evenly distributed, some workloads (e.g., *swaptions*) can shorten the cache raw lifetime by 20–30 %. If this is the case of future NVM caches, the system will eventually fail even when most of the NVM cells are still healthy. Therefore, it is extremely critical to design a write variation-aware cache management for the future success of NVM caches, and we need a cache policy that can tackle both inter-set and intra-set write variations.

[2] InterV and IntraV are not independent, but the worst-case variation can be modeled as the sum of them.

[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.

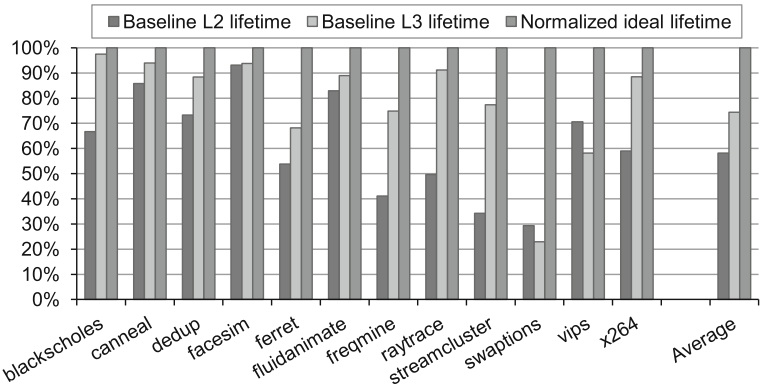

Fig. 9.3 The baseline lifetime of L2 and L3 caches normalized to the ideal lifetime (no write variations in the ideal case)

9.5 Starting from Inter-set Write Variations

9.5.1 Challenges of Inter-set Weal Leveling

Some wear-leveling techniques [12, 13, 15, 22] are focused on increasing the lifetime of NVM main memory. The key principle behind these techniques is to introduce an address remapping layer. This principle remains the same for cache inter-set wear leveling. However, there are some differences between designing wear-leveling policies for NVM main memory and NVM caches.

Using Data Movement: Main memory wear-leveling techniques usually use data movement to implement the address remapping. This is because we cannot afford to lose data in main memory and must move them to a new position after remapping. However, it is not free to move data from one location to another. First, we need temporary data storage to move the data. Second, one cache set movement involves multiple reads and writes. The cache port is blocked during the data movement, and the system performance is degraded. *Start-Gap* [12] is a recently proposed technique for main memory wear leveling. If we directly extend *Start-Gap* to handle the cache wear leveling, it falls into this category.

Using Data Invalidation: Another option to implement set address remapping for NVM caches is data invalidation. We can use cache line invalidations because we can always restore the cache data later from lower-level memories if it is not dirty. This special feature of caches provides us a new opportunity to design a low-overhead cache inter-set wear-leveling technique.

Compared to data movement, data invalidation needs no area overhead. In addition, to quantify the performance difference between these approaches, we design and simulate two systems (see Sect. 9.8 for detailed simulation settings). In the data movement system, the data in one cache set are moved to another after every 100

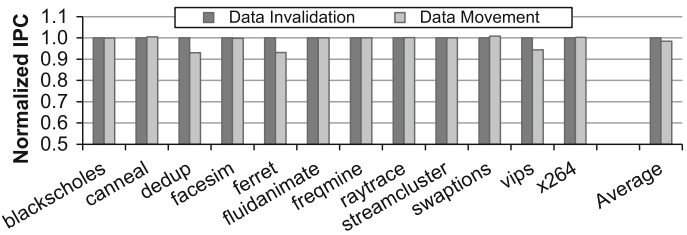

Fig. 9.4 The performance comparison between data invalidation and data movement

writes to the cache, which is extended from the start-gap technique [12]. On the other hand, the data are not moved but invalidated in the second system. Figure 9.4 shows the normalized performance of these two systems. Compared to the data invalidation system, the data movement system has worse performance (i.e., 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, we should notice that the latency of writing back and restoring data can often be hidden using write back buffer and MSHR techniques [10]. 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 enhance it by the Swap-Shift (SwS) scheme to reduce the inter-set write variation in NVM caches.

9.5.2 Swap-Shift

Considering data invalidation is more favorable in cache inter-set wear leveling, we modify the existing main memory wear-leveling technique and devise a new technique called SwS.

9.5.2.1 SwS Architecture

The basic concept behind SwS is to shift the location of each cache set. However, shifting all the cache sets at once brings a significant performance overhead. To solve this problem, SwS only swaps the data of two neighboring sets at once, and it can eventually shift all the cache sets by one location after $N - 1$ swaps, where N is the number of cache sets.

We use a global counter in SwS to store the number of cache writes, and we annotate it as Wr. We also use two registers, SwV (changing from 0 to $N - 2$) and shift value (ShV) (changing from 0 to $N - 1$), to track the current status of 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 [SwV] and set [SwV + 1] is triggered. Note that this swap operation only exchanges the set IDs and invalidates the data stored in these two sets (needs write-back if the data are dirty). After that, SwV is incremented by 1. One SwapR consists of $N - 1$ swaps and 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, SwV is reset to 0. One ShiftR consists of N shifts (i.e., SwapR).

Figure 9.5 is an example of how SwS shifts the entire cache by multiple swaps. It shows the SwV and ShV values during a complete ShiftR, which consists of 4 SwapR; it also shows 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 Sect. 9.10.1.

9.5.2.2 SwS Implementation

The implementation of SwS is shown in Fig. 9.6. We use a global counter to store wr and two registers to store ShV and SwV, respectively. When a logical set number (LS) arrives, the physical set number (PS) can be computed based on three different situations:

(1) If LS $=$ SwV, it means that this logical set is exactly the cache set should be swapped in this ShiftR. Therefore, PS is mapped to the current ShV.
(2) If LS $>$ SwV, it means that this set has not been shifted in this ShiftR. Therefore, PS is mapped to LS $+$ ShV.
(3) If LS $<$ SwV, it means that this set has been already shifted. Therefore, PS is mapped to LS $+$ ShV $+ 1$.

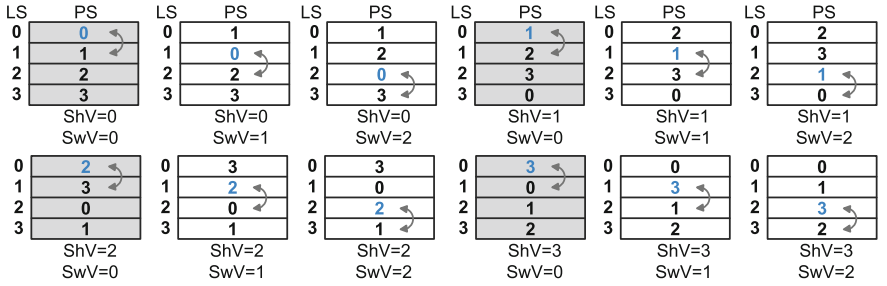

Fig. 9.5 One SwS ShiftR in a cache with 4 sets

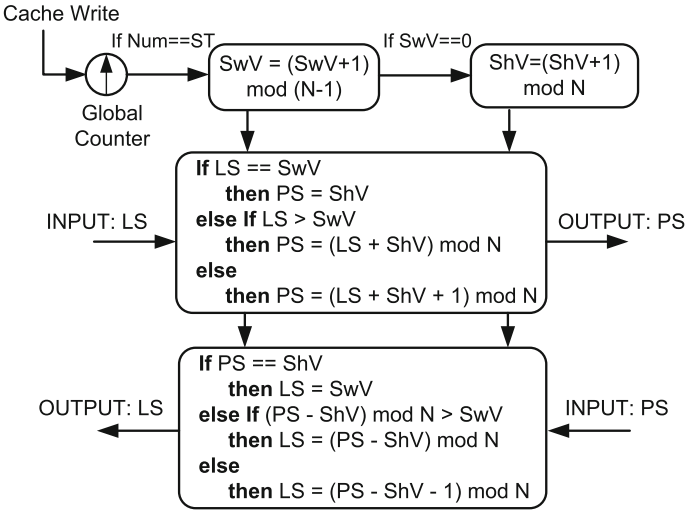

Fig. 9.6 The mapping between logical (LS) and physical set index (PS) in SwS

When a dirty cache line is written back to the next level of cache, the logical set address needs to be regenerated. The mapping from PS to LS is symmetric and is also given in Fig. 9.6. This mapping policy can be verified by the simple example in Fig. 9.5. Because SwV and ShV are changed along with cache writes, the mapping between LS and PS changes all the time. This ensures that the writes to different physical sets are balanced, reducing cache InterV.

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 synthesize the LS-to-PS address translation circuit in a 45 nm technology, and the circuit can handle a LS-to-PS translation within one cycle under a 3 GHz clock frequency.

9.6 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 Sect. 9.8.2 show that SwS alone cannot reduce intra-set variations. Therefore, in this section, we start with two straightforward techniques and then follow with a much improved technique, called PoLF, to tackle the cache intra-set variation problem.

9.6.1 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 is the frequent target of cache write hits and absorbs a large number of writes, its corresponding set must have a highly unbalanced write distribution.

In a traditional LRU cache policy, every accessed block is marked toward most recently used (MRU) 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 there is a cache write hit, the new data are 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 LF. Hence, the block containing the hot data have 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 the same performance concern explained in Sect. 9.5.1.

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 have 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.

9.6.2 Hot Set Line Flush

One of the simplest solutions is to add counters to cache tags, storing the write count of each cache line. 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, we detect a very hot data and should flush it. In this way, the hottest line can be replaced with other data, and the hot data can be reloaded into a relatively cold cache line. The architecture of HoLF is shown in Fig. 9.7.

However, HoLF has significant overhead in both area and performance. The area overhead is large since it requires one counter for every cache line. Considering the typical cache line is 64-byte wide and assuming the write counter is 20-bit, the hardware overhead is more than 3.7 %. In addition, 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.

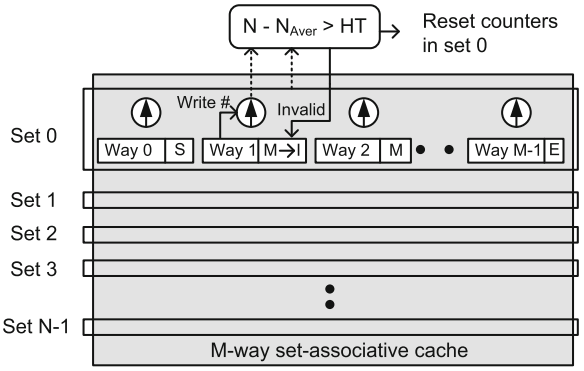

Fig. 9.7 The cache architecture of HoLF. The counters to store the write count are added to every cache line

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

9.6.3 Probabilistic Set Line Flush

The motivation of PoLF is to flush hot data probabilistically instead of deterministically.

9.6.3.1 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 are, 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 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.

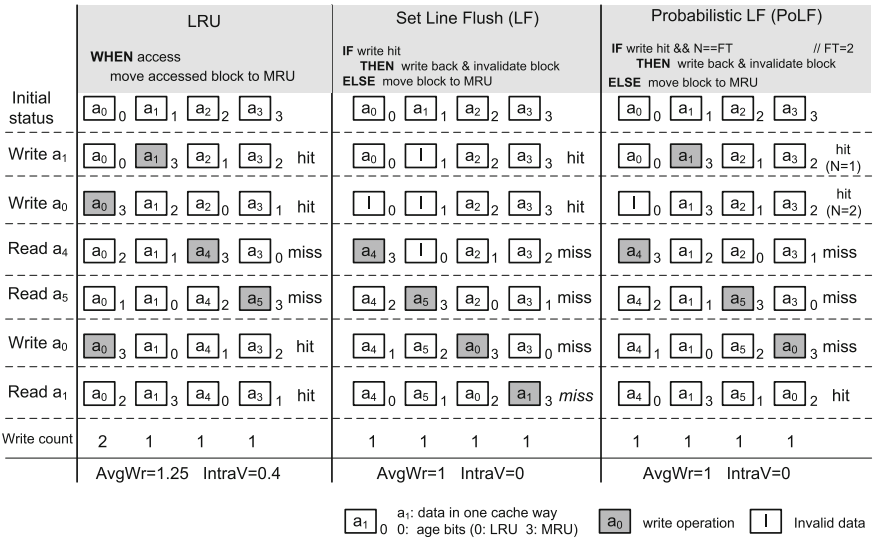

Fig. 9.8 The behavior of one cache set composed of 4 ways under LRU, LF, and PoLF polices for the same access pattern. The total write count of each cache way, the average write count and the intra-set variation are marked, respectively

Comparison with Other Policies: Figure 9.8 shows the behavior of a 4-way cache set under LRU, LF, and PoLF polices for an exemplary access pattern. Our observations from this example are:

(1) For LRU, the hot data a_0 are moved to the MRU position (age bits = 3) after each write hit and are never replaced by other data. Thus, the intra-set variation using LRU is the largest one among all the polices.
(2) 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 are reloaded into another cache line. However, data a_1 are also flushed since every write hit causes one cache line flush, and it brings one additional access miss.
(3) 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 are moved to another cache line. In addition, compared to LF, the number of misses is reduced because a_1 is not flushed.

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

[4] FT is set as 2 for this illustration. More typical values are much larger and shown in Sect. 9.8.

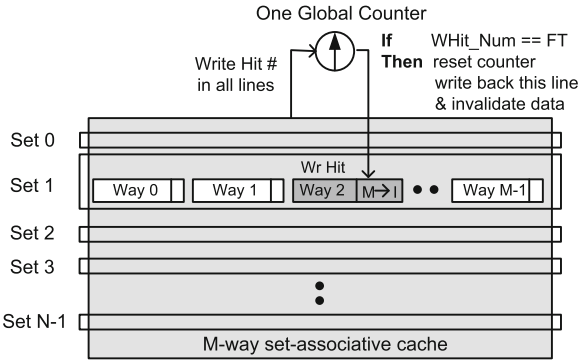

Fig. 9.9 The cache architecture of PoLF. Only one global write hit counter is added to the entire cache

9.6.3.2 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 9.9 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 FT.

9.7 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.

9.8 Experiments

In this section, we first describe our experiment methodology, and 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 NVM cache lifetime.

9.8.1 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 [1] benchmark workloads.[5] 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 [2] 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 9.1, 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 NVM technologies. The system parameters are given in Table 9.2.

9.8.2 Write Variations Reduction

9.8.2.1 Effect of SwS on Inter-set Variations

In SwS, the inter-set variation reduction is related to the number of ShiftR during the experimental time. Assuming there are N sets in the cache, one ShiftR includes N SwapR and one SwapR has $N-1$ swaps. After each ShiftR, all the cache sets are

Table 9.1 Workload characteristics in L2 an L3 caches under our baseline configuration

Workload	L2 cache		L3 cache	
	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

[5] A supplementary experiment on multi-program workloads is given in Sect. 9.9.3.

Table 9.2 Baseline configurations

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

* We envision L1 is still SRAM due to performance concerns and NVM write endurance limits

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.

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

$$\text{RRN} = \frac{W_{\text{total}}}{\text{ST} \times N \times (N-1)} = \frac{\text{WPI} \times I_n}{\text{ST} \times N \times (N-1)} \tag{9.6}$$

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 9.10 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 1 MB 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 ST can be set larger than 100,000 according to Eq. 9.6 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 9.10b 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.

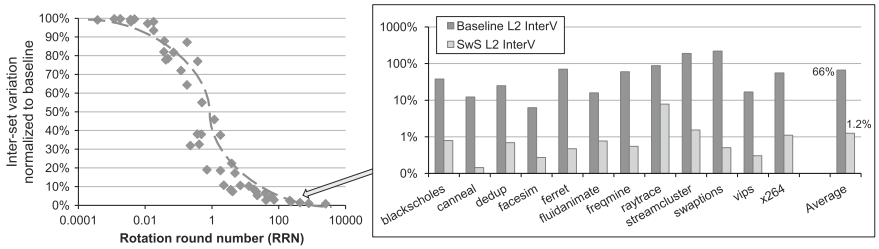

Fig. 9.10 Inter-set variations normalized to baseline when RRN increases in SwS scheme. The zoom-in sub-figure shows the detailed L2 inter-set variation of different workloads after adopting the SwS scheme when RRN equals to 100

9.8.2.2 Effect of PoLF on Intra-Set Variations

Figure 9.11 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 9.11 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.

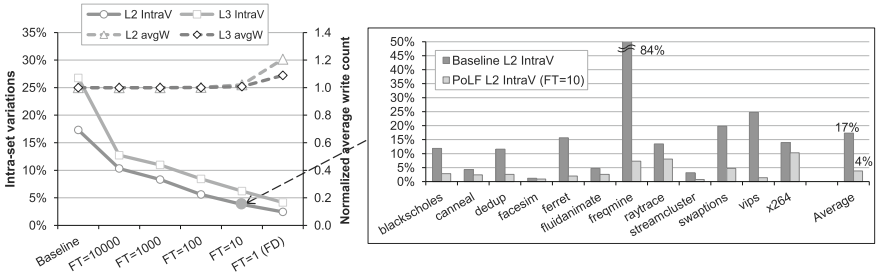

Fig. 9.11 The average intra-set variation and the average write count normalized to baseline for L2 and L3 caches after adopting a PoLF scheme. The zoom-in sub-figure shows the detailed L2 intra-set variation for different workloads after adopting an PoLF scheme with line flush threshold (FT) of 10

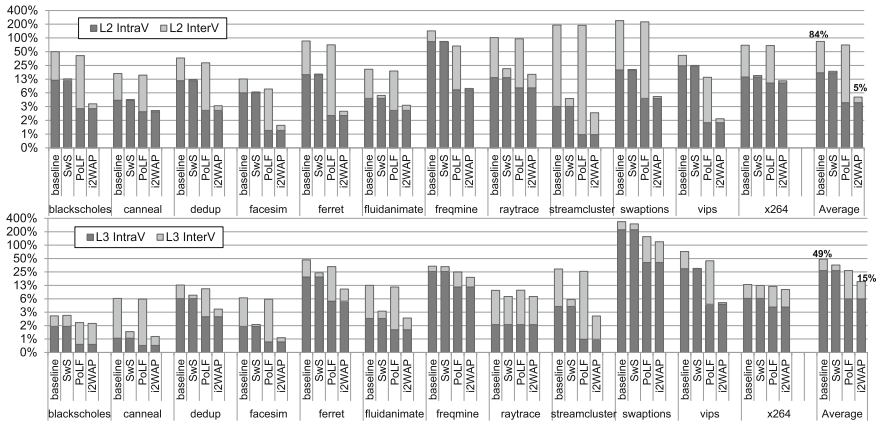

Fig. 9.12 The total variation for L2 and L3 caches under the baseline configuration, SwS scheme (RRN = 100), PoLF scheme (FT = 10) and i^{2}WAP policy. Each value is broken down to the inter-set variation and the intra-set variation. Note that a log scale is used to cover a large range of variations

9.8.2.3 Effect of i^{2}WAP on Total Variations

Figure 9.12 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^{2}WAP reduces the total variations significantly. Figure 9.12 shows that on average the total variation is reduced from 84 to 5 % for L2 caches and from 49 to 15 % for L3 caches.

9.8.3 Lifetime Improvement of i^{2}WAP

After reducing inter-set and intra-set variations, the lifetime of NVM caches can be improved. Figure 9.13 shows the LI of L2 and L3 caches after adopting SwS only, PoLF only, and the combined i^{2}WAP policy, respectively. The LI varies based on the workload. Basically, the larger the original variation value is, the bigger the improvement a workload has. The overall LI is 75 % (up to 224 %) for L2 caches and 23 % (up to 100 %) for L3 caches.

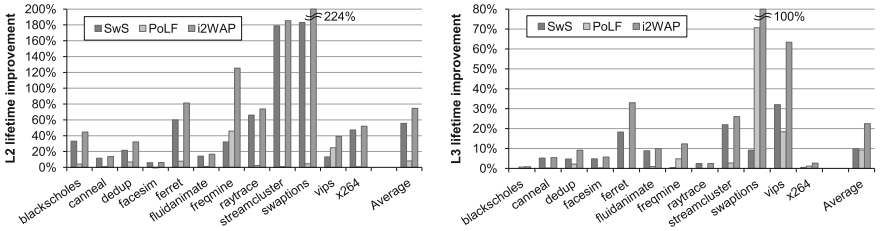

Fig. 9.13 The lifetime improvement after adopting i^{2}WAP using Eq. 9.5 (*Left* L2, *Right* L3)

9.9 Sensitivity Study

9.9.1 Sensitivity to Cache Associativity

As shown in Table 9.2, we use 8-way associative L2 and L3 caches in the baseline system. To study on the effectiveness of i^{2}WAP on different cache configurations, we do a sensitivity study on different associativity numbers ranging from 4 to 32. All the other system parameters remain the same.

Figure 9.14 shows the total variations of L2 and L3 caches under different policies when the associativity is changed. For both L2 and L3 caches in the baseline system, with the increase in the cache associativity, InterV is decreased and IntraV is increased. The reason is that when the cache capacity is fixed, the set number decreases as the associativity increases. Thus, more writes are merged into one cache set and the write variation between different sets becomes smaller. Furthermore, IntraV is amplified since the number of cache block in one set is increased and the write imbalance becomes worse.

Regardless of how the associativity changes, adopting i^{2}WAP reduces the total variations significantly by combining SwS and PoLF. Figure 9.14 shows that on average the total variation of the 4-way system is reduced from 109 to 17 % for L2

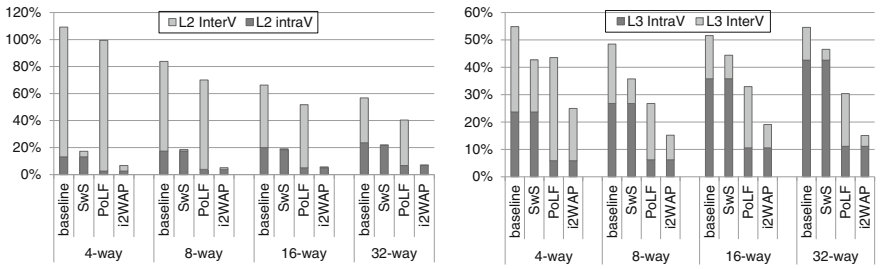

Fig. 9.14 The total variation for L2 and L3 caches with 4-way, 8-way, 16-way, and 32-way under the baseline configuration, SwS scheme (RRN = 100), PoLF scheme (FT = 10), and i^{2}WAP policy. Each value is broken down to the inter-set variation and the intra-set variation

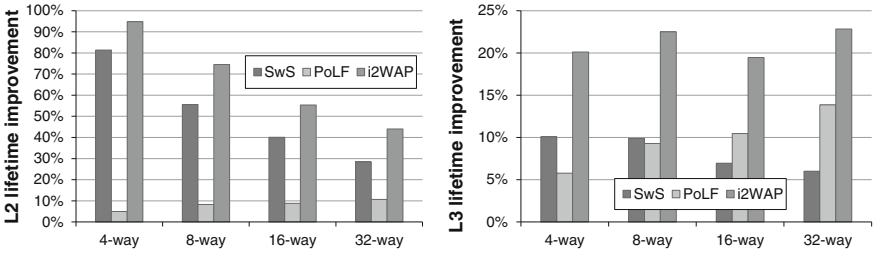

Fig. 9.15 The lifetime improvement after adopting i^{2}WAP with different cache associativity using Eq. 9.5 (*Left* L2, *Right* L3)

caches and from 55 to 25 % for L3 caches; for the 16-way system, the total variation is reduced from 66 to 6 % for L2 caches and from 52 to 19 % for L3 caches; for the 32-way system, the total variation is reduced from 57 to 7 % for L2 caches and from 55 to 15 % for L3 caches.

The NVM cache lifetime is improved after reducing InterV and IntraV. Figure 9.15 shows the LI of L2 and L3 caches after adopting SwS only, PoLF only, and the combined i^{2}WAP policy, respectively. On average, the LI is 95 and 20 % for L2 and L3, respectively, in a 4-way system; it is 55 and 20 % for L2 and L3, respectively, in a 8-way system; it is 44 and 23 % for L2 and L3, respectively, in a 16-way system. In general, the larger the original variation value is, the bigger the improvement i^{2}WAP can bring.

9.9.2 Sensitivity to Cache Capacity

Another sensitivity study is targeted to the cache capacity. In Sect. 9.8, we use 1 MB L2 and 8 MB L3 caches as shown in Table 9.2. We expect that i^{2}WAP also works effectively on caches with different capacity. We have experiments on different L2 capacity ranging from 512 KB to 4 MB and different L3 capacity ranging from 4 to 32 MB. Figure 9.16 shows the result. On average the total variation is reduced by 90–95 % for L2 caches and 58–73 % for L3 caches, respectively.

Figure 9.17 is the according result of the LI. On average, the lifetime improvement is 66–153 % and 22–26 %, respectively. These results validate that i^{2}WAP works effectively regardless of the cache capacity. For L2 caches, we can see that as capacities increase, the value of LI also increases. The reason is that the write imbalance is worse in larger capacity caches and the baseline variation value is also increased. Thus, larger caches provide more space for i^{2}WAP to decrease the variation and improve the lifetime. For L3 caches, the trend of the variation growth is much smaller than the ones in L2 caches, and the intra-set variations occupy a larger proportion in the total baseline variations. Thus, the LI of L3 caches when the capacity is changed shows smaller differences than L2 caches.

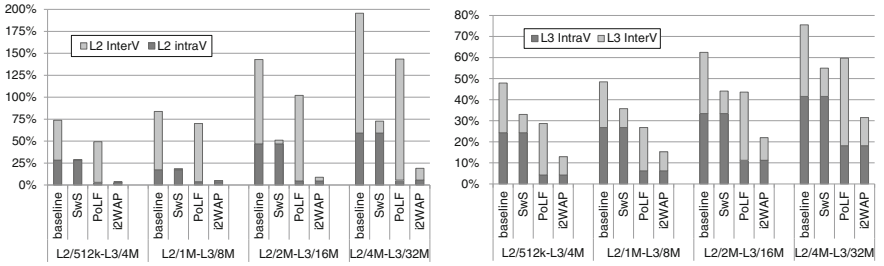

Fig. 9.16 The total variation for L2 and L3 caches with different capacities under the baseline configuration, SwS scheme (RRN = 100), PoLF scheme (FT = 10), and i^{2}WAP policy. Each value is broken down to the inter-set variation and the intra-set variation

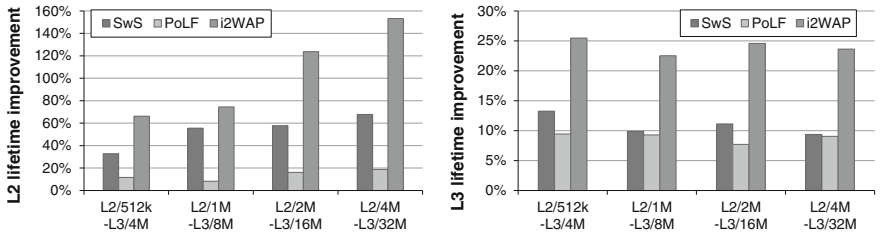

Fig. 9.17 The lifetime improvement after adopting i^{2}WAP with different cache capacities using Eq. 9.5. (*Left*: L2, *Right*: L3)

9.9.3 Sensitivity to Multi-program Applications

All the previous simulations are based on multi-thread workloads. To study the effectiveness of i^{2}WAP on multi-program applications, we simulate workload mixtures from SPEC CPU2006 benchmark suite [19]. The other experimental configurations are the same as described in Sect. 9.8.

Figure 9.16 shows the variations of L2 and L3 caches for multi-program applications using mixed SPEC CPU2006 workloads in a 4-core system. Table 9.3 lists the combination of the workload mixtures. Intuitively, multiple cores share one L3 cache and run different programs, and the access traffic to the L3 cache should be well mixed and thus balanced. However, Figure 9.16 shows that both InterV and IntraV in the shared L3 cache are still very large in some cases. On average, InterV and IntraV of L3 caches are 28 % (up to 100 %) and 26 % (up to 64 %), respectively. Similar to the results of multi-thread experiments, the variations in L2 caches is larger than the ones in L3 caches since L2 only serves one program and has more inbalanced write. On average, the total variation is reduced from 132 to 24 % for L2 caches and from 54 to 15 % for L3 caches, respectively (Fig. 9.18).

Figure 9.19 shows the LI for the multi-program workloads. For L2 and L3 caches, the overall LI is 88 % (up to 387 %) and 33 % (up to 136 %), respectively. Therefore, i^{2}WAP is effective in reducing the write variations and improving the cache lifetime for multi-program workloads.

Table 9.3 The workload list in the mixed groups

Mixed group	Workloads
Mix 1	astar+bwaves+bzip2+gcc
Mix 2	bzip2+astar+gobmk+h264ref
Mix 3	gromacs+bzip2+gcc+gobmk
Mix 4	gcc+gromacs+hmmer+namd
Mix 5	gobmk+h264ref+hmmer+gromacs
Mix 6	h264ref+hmmer+milc+namd
Mix 7	milc+namd+omnetpp+wrf
Mix 8	namd+h264ref+gcc+astar
Mix 9	omnetpp+wrf+astar+bwaves
Mix 10	wrf+milc+bwaves+gromacs

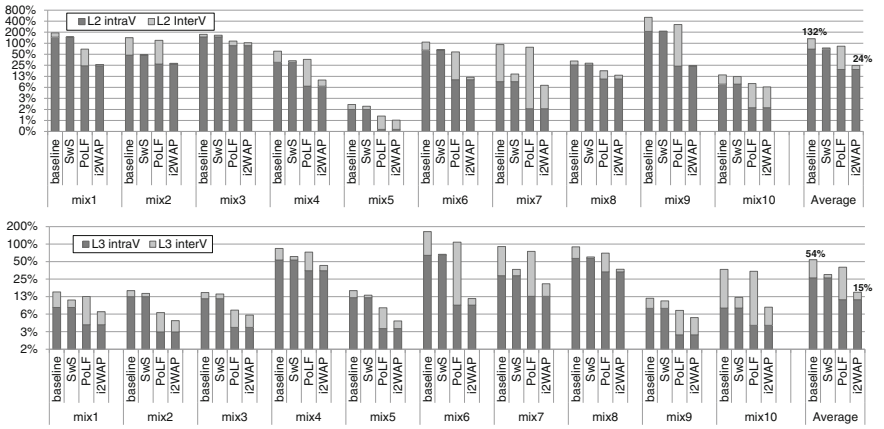

Fig. 9.18 The total variation for L2 and L3 caches for multi-program applications using mixed SPEC CPU2006 workloads under the baseline configuration, SwS scheme (RRN = 100), PoLF scheme (FT = 10), and i^{2}WAP policy. Each value is broken down to the inter-set variation and the intra-set variation

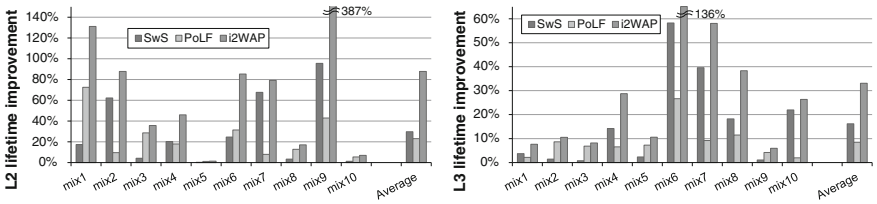

Fig. 9.19 The lifetime improvement after adopting i^{2}WAP for multi-program applications using Eq. 9.5. (*Left* L2, *Right* L3)

9.10 Analysis of Other Issues

9.10.1 Performance Overhead of i²WAP

Since i²WAP causes extra cache invalidations, it is necessary to compare its performance to a baseline system without wear leveling.[6] Figure 9.20 shows the performance overhead of a system in which L2 and L3 caches using i²WAP with ST = 100,000 and FT = 10 compared to a baseline system in which an LRU policy is adopted.

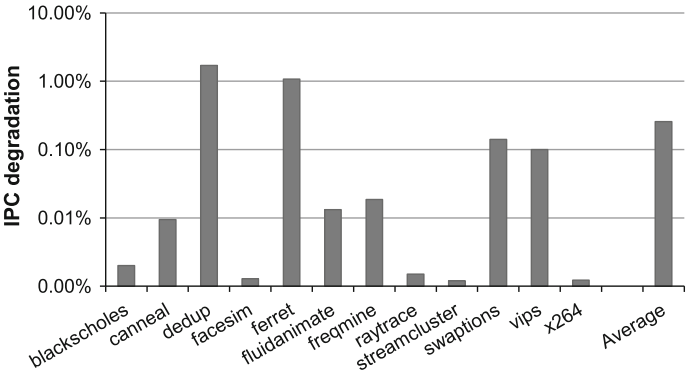

Fig. 9.20 The system IPC degradation compared to the baseline system after adopting the i²WAP policy

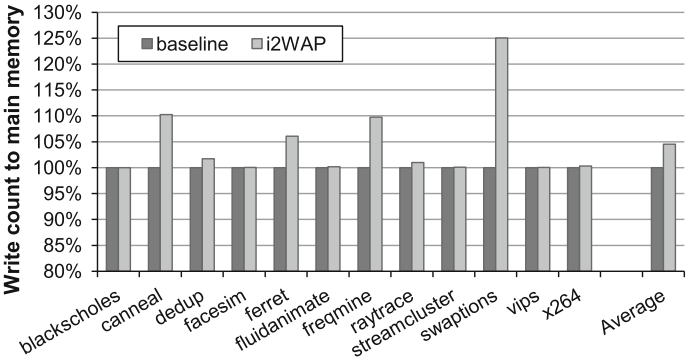

Fig. 9.21 The write count to the main memory normalized to the baseline system after adopting i²WAP

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

As shown in Fig. 9.20, 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 remapped for each operation.
- In PoLF, write hit accesses are infrequent enough to ensure the frequency of LF operations is low (e.g., once per 10^5 instructions), and only one cache line is flushed each time. In addition, designers can trade-off between the number of flush operations and the variation value by adjusting FT.

9.10.2 Impact on Main Memory

We also evaluate the impact of extra write backs on main memory. Figure 9.21 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 than that of caches, the endurance requirement for nonvolatile 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.

9.10.3 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 Sect. 9.1). 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 [6, 14–16, 21]. 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 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

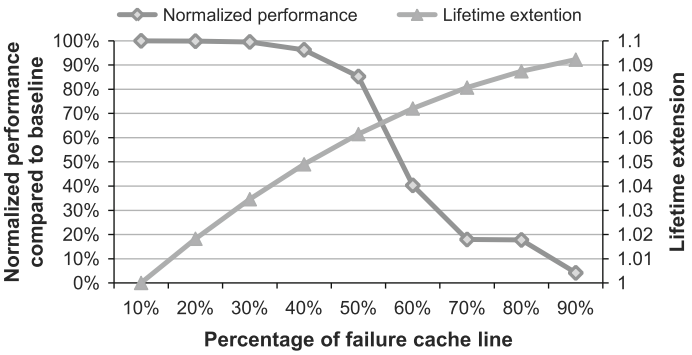

Fig. 9.22 The performance degradation and lifetime extension during gradual cache line failure on a nonvolatile cache hierarchy

is reduced. Figure 9.22 shows an analysis of an ReRAM-based cache hierarchy with 32 KB L1 caches, 1 MB L2 caches, and 8 MB 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 %.[7]

9.10.4 Security Threat Analysis

Thus far, the workloads we have considered are only typical. However, memory technologies with limited write endurance always pose a unique security threat. An adversary might design an attack that stresses a few lines in the cache to reach the endurance limit and then cause the system fail.

The security threat models for main memory and caches are different. The cache address space is a subset of the total memory space, and the result of cache replacement policies is difficult to predict accurately. This makes random attacks, such as birthday paradox attack [17] difficult to implement in caches. Thus, a more effective attack for nonvolatile caches is repeated address attack (RAA) [12].

For a simple RAA, the attacker can write a data line in cache repeatedly and then cause cache line failure. It is easy to implement in a cache without an endurance-aware policy. For L1 data caches, when a cache line is repeated written, it would not be replaced under an LRU policy since it is continuously referenced and the address of the attacked line is never changed during the repeated writing. The mechanism is similar in a lower-level cache, and the only difference is that the attacker would use the higher-level cache's write-back to repeatedly write the lower-level cache. For example, circularly writing $\text{NUM}_{\text{ways}} + 1$ of data in one set of L1 data caches would force it to write back to the L2 cache repeatedly. Assuming it takes 10–50 cycles to

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

write back to one L2 cache line, then the time required to make a L2 cache line fail is given by

$$\text{Time to Fail} = \frac{\text{Cycle per write} \times \text{Cell endurance}}{\text{Cycles per second}} = 6\text{–}30\,\text{min} \tag{9.7}$$

Thus, the traditional cache policy opens a serious security problem for nonvolatile caches with limited write endurance.

However, our proposed i^{2}WAP policy can mitigate this problem. By adopting i^{2}WAP, the injected attacks would be distributed in every different cache line because of the following two reasons:

(1) The mapping between physical sets and logical sets is shifted by the SwS policy and according to the write count, which is hard to predict. In addition, it becomes more difficult for the attacker to guess the shift period if there are other processes competing the same cache resource.
(2) Cache line invalidation is quickly triggered by repeatedly write hits. PoLF guarantee a high probability to invalidate the attacked cache line, and data are then loaded in another random location, and the attacker cannot predict the new location.

Thus, the attacks would be distributed in every different cache line. For a 1 MB L2 cache with i^{2}WAP, the time to make a cache line fail would be

$$\begin{aligned}\text{Time to Fail} &= \frac{\text{Number of cache line} \times \text{Cycle per write} \times \text{Cell endurance}}{\text{Cycles per second}} \\ &= 2\text{–}10\ \text{months}\end{aligned} \tag{9.8}$$

In lower-level caches with a larger number of cache lines and longer write cycles, the time to attack the system till failure is even longer. Thereby, such a long duration is sufficient to detect the abnormal attack accesses and make the system safe.

9.11 Conclusion

Modern computers require larger memory system, but the scalability of traditional SRAM and DRAM is constrained by leakage and cell density. Emerging NVM is a promising alternative to build large main memory and on-chip caches. However, NVM technologies usually have limited write endurance, and the write variation would cause heavily written memory blocks to fail much earlier than their expected lifetime. Thus, wear-leveling techniques are studied to eliminate memory write variations.

In this chapter, we use i^{2}WAP as an example to show how to design an 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 NVM on-chip L2 and L3 caches. The implementation overhead of i^2WAP is small, only needing two global counters and two global registers, but it can improve the lifetime of NVM caches by 75 % on average and up to 224 % over the conventional cache management policies.

References

1. Bienia, C., Kumar, S., Singh, J. P., & Li, K. (2008). The PARSEC benchmark suite: Characterization and architectural implications. In PACT. pp. 72–81.
2. Binkert, N., et al. (2011). gem5: A multiple-ISA full system simulator with detailed memory model. Computer Architecture News.
3. Chen, B., Lu, Y., Gao, B., et al. (2011). Physical mechanisms of endurance degradation in TMO-RRAM. 12.3.1-12.3.4. http://dx.doi.org/10.1109/IEDM.2011.6131539.
4. Huai, Y. (2008). Spin-transfer torque MRAM (STT-MRAM): Challenges and prospects. *Association of Asia Pacific Physical Societies, Bulletin*, *18*(6), 33.
5. International Technology Roadmap for Semiconductors. (2012). Process integration, devices, and structures 2012 update. http://www.itrs.net/.
6. Ipek, E., Condit, J., Nightingale, E. B., Burger, D., & Moscibroda, T. (2010). Dynamically replicated memory: Building reliable systems from nanoscale resistive memories. *ASPLOS*, 3–14.
7. Joo, Y., Niu, D., Dong, X., Sun, G., Chang, N., & Xie, Y. (2010). Energy- and endurance-aware design of phase change memory caches. In Automation and Test in Europe Conference and Exhibition, Design, pp. 136–141.
8. Kim, K., & Ahn, S. J. (2005). Reliability investigations for manufacturable high density PRAM. In IRPS, pp. 157–162.
9. Kim,Y.-B., Lee, S. R., Lee, D., Lee, C.B., et al. (2011). Bi-layered RRAM with unlimited endurance and extremely uniform switching. In VLSI. pp. 52–53.
10. Kroft, D. (1998). Lockup-free instruction fetch/prefetch cache organization. In *25 Years of the International Symposia on Computer architecture (selected papers)*, pp. 195–201.
11. Lin,W. S., Chen, F. T., Chen, C. H. L., & Tsai, M.-J.. (2010). Evidence and solution of over-RESET problem for HfO_x based resistive memory with sub-ns switching speed and high endurance. In *Proceedings of the International Electron Devices Meeting*. 19.7.1–19.7.4.
12. Qureshi, M. K., Karidis, J. P., et al. (2009a). Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling. In *Proceedings of the International Symposium on Microarchitecture*, pp. 14–23.
13. Qureshi, M. K., Srinivasan, V., & Rivers, J. A. (2009b). Scalable high performance main memory system using phase-change memory technology (pp. 24–33). Architecture: In *Proceedings of the International Symposium on Computer*.
14. Schechter, S., Loh, G. H., Straus, K., & Burger, D. (2010). Use ECP, not ECC, for hard failures in resistive memories (pp. 141–152). In *Architecture: Proceedings of the International Symposium on Computer*.
15. Seong, N. H., Woo, D. H., & Lee, H.-H. S. (2010a). Security refresh: Prevent malicious wear-out and increase durability for phase-change memory with dynamically randomized address mapping (pp. 383–394). In *Architecture: Proceedings of the International Symposium on Computer*.
16. Seong, N. H., Woo, D. H., Srinivasan, V., Rivers, J. A. & Lee, H.-H. S. (2010b). SAFER: Stuck-at-fault error recovery for memories (pp. 115–124). In *Proceedings of the International Symposium on Microarchitecture*.

17. Seznec, A. (2010). A phase change memory as a secure main memory. *Computer Architecture Letters*, *9*(1), 5–8.
18. Slaughter, J. M., Rizzo, N. D., Mancoff, F. B., Whig, R., Smith, K., Aggarwal, S., et al. (2010). Toggle and spin-torque MRAM: Status and outlook. *Journal of the Magnetic Society of Japan*, *5*, 171.
19. Spec, CPU. (2006). SPEC CPU2006. http://www.spec.org/cpu2006/.
20. Wang, J., Dong, X., Xie, Y., & Jouppi, N. P. (2013). i^2WAP: Improving non-volatile cache lifetime by reducing inter- and intra-set write variations. In *Proceedings of the International Symposium on High-Performance Computer Architecture*.
21. Yoon, D. H., Muralimanohar, N., Chang, J., Ranganathan, P., et al. (2011). FREE-p: Protecting non-volatile memory against both hard and soft errors (pp. 466–477). In *Proceedings of the International Symposium on High-Performance Computer Architecture*.
22. Zhou, P., Zhao, B., Yang, J., & Zhang, Y. (2009). A durable and energy efficient main memory using phase change memory technology (pp. 14–23). In *Architecture: Proceedings of the International Symposium on Computer*.