International Journal of Multimedia and Ubiquitous Engineering
Vol.11, No.1 (2016), pp.293-310
http://dx.doi.org/10.14257/ijmue.2016.11.1.28

A Survey on Phase Change Memory-Aware Cache Management

Kaimeng Chen[1,2], Peiquan Jin[1,2*] and Lihua Yue[1,2]

[1]*School of Computer Science and Technology, University of Science and Technology of China, 230027, Hefei, China*
[2]*Key Laboratory of Electromagnetic Space Information, Chinese Academy of Sciences, Hefei 230027, China*
jpq@ustc.edu.cn

Abstract

PCM (phase change memory) has been regarded as one of the most promising NVRAM. Due to its excellent features including high access speed, good scalability, byte addressability, and low idle power, PCM is expected to be an alternative to DRAM or flash memory in the future. So far, many different kinds of architecture have been proposed to merge PCM into the memory hierarchy of computer systems. For PCM-based computer systems, considering that PCM has various properties differing with DRAM and conventional storage media, traditional cache management policies based on DRAM and disks have to be re-designed. In this paper, we present a survey on the cache management policies for PCM-based systems that employ PCM as main memory or as secondary storage. Specially, for different PCM-based architectures, we explain how PCM-aware cache management policies exploit PCM's advantages and overcome PCM's drawbacks like asymmetric read/write latencies.

Keywords: *Phase change memory, DRAM, cache management policy*

1. Introduction

Recently, with the increase of data scale in modern servers and application, the problem of the gap between high speed CPU and slow I/O access speed storage become more serious. Conventionally, DRAM is used as main memory to cache data from storage and bridge the gap between CPU and storage. But in the face of nowadays large data scale and large capacity memory requirement, DRAM suffers from its limited scalability and high energy consumption. Therefore, new memory technology that has low access latency, high density and low energy consumption become an urgent need. Under the circumstance, non-volatile memory technologies gain extensive attention of industry and academia.

Non-volatile memory is a new kind of storage medium aiming to solve the problem due to the disadvantages of DRAM and disk. The characteristics of non-volatile memory are persistent, byte-addressability, fast access speed and high density. Compared with disk or flash memory, the access latency of non-volatile memory is much lower. Compared with DRAM, non-volatile memory has advantages of scalability and no leakage energy.

Some non-volatitle memory technologies have already emerged and develop rapidly in recent years, such as MRAM, STT-RAM, RRAM, FeRAM and PCM. Among these technologies, due to its performance and potential of access latency and density, PCM is considered to be a promising candidate to substitute DRAM [1-6]. Table 1 shows the comparison between DRAM and PCM.

PCM has many excellent properties: the read performance of PCM is similar to that of DRAM; PCM's density is expected to be considerably large because its feature size can be very small and each PCM cell can store multiple bits [2-3, 6-8]; and PCM has low idle energy consumption. But PCM also suffer from two critical disadvantages: first, PCM

ISSN: 1975-0080 IJMUE

has read-write cost asymmetry problem, access latency and energy consumption of write operation are much higher than that of read operation; second, PCM has endurance problem, each PCM cell can only be set and reset for limited times (about 10^8).

Considering PCM's unique performance profile, PCM can be merged into different hierarchy of current computer system. First, PCM can be merged into main memory to completely replace DRAM and build non-volatile main memory system [3, 5, 9]. Second, PCM can be used in combination with DRAM to build hybrid memory system [4, 10-14]. Final, there are also some studies based on using PCM as storage device to replace disk and flash SSD [15-19].

Merging PCM into conventional computer system brings new problems for cache management at the corresponding memory levels. Traditional cache management policies [20-27] are based on conventional CPU cache-DRAM-disk architecture. In conventional architecture, there is a great gap of access latency and capacity between different memory hierarchies, and the write cost and read cost are equal at each level. So traditional cache management policy focus on improving the hit ratio of the cache. But PCM has write endurance problem and asymmetric read/write problem, so PCM-based cache management policy should not consider hit ratio only. Reducing PCM write overhead to increase PCM's lifetime and improve the overall performance should also be taken into account.

There are already some studies that summarize the technologies designed for PCM-based system [28-30]. But these studies do not pay special attention to cache management problems on PCM system. In this paper, we focus on surveying PCM-aware cache management policies for different PCM-based system architectures. The remainder of this survey is organized as follow. Section 2 introduces the different types of PCM system architecture and the new challenges to cache management policy. Section 3 summarizes the cache management policies for last-level-cache based on PCM main memory and hybrid memory. Section 4 describes the design and management of DRAM cache for PCM main memory. Section 5 presents page management and replacement algorithms on PCM main memory and hybrid memory. Section 6 presents the studies on DRAM main memory management policies for PCM storage. Finally, Section 7 presents the conclusion of this paper.

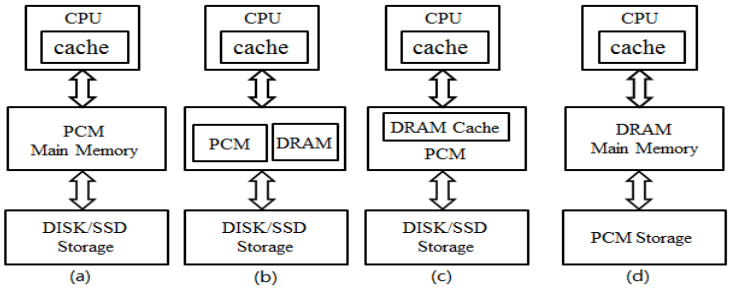

Figure 1. Different PCM System Architecture

2. PCM System Architecture

In this section, we introduce different types of PCM system architectures that merge PCM into different memory hierarchy and adopt PCM in different ways. For each type of architecture, we discuss how PCM brings changes to conventional computer system architecture and challenges to cache management policies of the corresponding memory levels.

2.1 Pure PCM Main Memory

To satisfy the requirement of big capacity main memory system, PCM is expected to be a substituent of DRAM due to PCM's great scalability, so the PCM system architecture that replace DRAM totally with PCM is proposed [3, 5, 9], as shown in Figure 1(a). Compared with DRAM, PCM main memory can have much larger memory capacity and low idle power consumption. But consider the much higher write latency of PCM, replacing DRAM with PCM can still degrade the overall performance. So many studies are proposed to reduce the write operation cost of PCM main memory [5, 31-36].

PCM main memory also bring challenges to DRAM-based cache management policies for last-level cache (LLC) and main memory, due to PCM's asymmetric read/write cost problem and limited endurance problem that do not exist in DRAM.

LLC management: Based on DRAM main memory, to mitigate the access latency gap between CPU cache and the main memory, cache management policies of LLC focus on identifying cache blocks with high likely to be re-referenced and keep those blocks in cache to improve hit ratio [26, 37, 38]. But based on PCM main memory, LLC management problem is similar to DRAM cache management problem for flash SSD storage [63-68]. Due to PCM's poor write performance, to improve the overall performance, the cache management policy should consider not only the hit ratio but also reducing write-backs from LLC to PCM.

Main memory management: cache management policies for DRAM do not need to consider the location of the pages since DRAM does not have the endurance problem. For PCM main memory, due to the limited write count on each PCM cell, so the policy should care about the physical address when a page is cached into PCM from storage or a page in PCM is replaced with one other page from storage, in order to prevent wearing PCM partial regions too much.

2.2 PCM Main Memory with DRAM Cache

The poor write performance of PCM main memory significantly degrades the overall performance of system. To achieve a better main memory system with large capacity and low latency, the DRAM cache architecture that adopts PCM as main memory and adopts DRAM as cache for PCM has been proposed [4, 10, 11], as shown in Figure 1(b) DRAM cache is an in-house cache which is managed by memory controller and hidden to the operation system. By serving write accesses in DRAM cache, PCM main memory gets low write latency and increased lifetime.

Since OS is not aware of DRAM cache, cache management policies of LLC and main memory can still regard the DRAM cache architecture as PCM main memory. But the cache management policy of DRAM cache should be specially designed. Due to PCM's asymmetric read/write cost, serving a write access in DRAM cache benefits main memory performance more than serving a read access in DRAM cache, so the policy for DRAM cache should consider the tradeoff between write access hit ratio and overall hit ratio of the cache.

2.3 Hybrid DRAM and PCM Memory Architecture

To exploit the advantages of DRAM and PCM, hybrid DRAM and PCM memory architecture that adopts both DRAM and PCM as main memory is proposed [12-14], as shown in Figure 1(c). Hybrid PCM and DRAM memory manages PCM space and DRAM space together as main memory space, and different types of space can be identified according to physical address and managed by the OS.

Unlike traditional main memory system that all the space has uniform access time, hybrid DRAM and PCM memory architecture has heterogeneous memories with different access time. Cache management policies for hybrid memory must be aware of the difference between PCM and DRAM.

For LLC management, consider that DRAM-based LLC management and PCM-based LLC management have different requirements, the policy for LLC should manage data from DRAM and data from PCM in different ways, and properly partition cache size for DRAM data and PCM data, to take both the hit ratio and write-backs to PCM into account.

For main memory management, similar to data allocation between HDD and SSD for hybrid storage architecture [69-71], page allocation between PCM and DRAM significantly affect the overall performance of hybrid memory architecture. In order to efficiently exploit DRAM to hide PCM's poor write performance, the policy for hybrid main memory should be able to predict page's access pattern, and allocate write-intensive pages to DRAM to reduce write access count on PCM.

2.4 PCM Storage

Compared with flash memory, PCM performs much better on access time and reliability. The cost per bit of PCM is promising to be approach the cost of disks [39] and the density of PCM is expected to reach the storage level [40]. So PCM is also considered as one storage candidate and an alternative to flash memory, as shown in Figure 1(d). Since PCM storage has the main-memory-level access time and storage-level capacity, it blurs the strict boundaries between memory and storage.

As shown in Figure 1(d), DRAM is still the main memory for PCM storage. Since the read access latency of PCM is similar to that of DRAM, the cost of missing a page in DRAM is not unacceptable, but the cost that caching a page in DRAM need to be considered if the cached page is clean not re-referenced in DRAM before evicting. Consider PCM's write cost and endurance, the cache management policy for DRAM need to keep hot-dirty pages in PCM and improve the hit ratio of write accesses to reduce write-backs to PCM from evicting dirty pages.

3. Last-level Cache Management Technology

3.1 Last-Level Cache Management Based on PCM Main Memory

When a dirty cache block in the LLC is evicted, the block should be written back to the main memory. For PCM main memory that suffers from the long write latency and high write energy cost, evicting dirty cache blocks frequently can degrade the overall performance and increase energy consumption of main memory system. So PCM-aware cache management policies of LLC should consider the different cost of evicting a clean block and evicting a dirty block to balance the tradeoff between the hit ratio of LLC and the write overhead of PCM, for the overall performance of the system. Since there are also some cache management policies

Zhang *et al*. [41] present a read-write aware replacement algorithm RWA and its improved version I-RWA for PCM main memory. RWA and I-RWA are designed to manage LLC, and DRAM cache if PCM main memory adopts it. These algorithms are based on RRIP [26]. Like RRIP, RWA set a Re-reference Prediction Value (RRPV) to each cache block. For *n*-way cache, when a read request misses in the cache, RWA set RRPV of the new data line to *n*-2; when a write-back from the upper-level cache misses, RWA set RRPV of the new data line to 0. Once a data line hits in the cache, its RRPV is reset to 0. To select a victim to evict, RWA randomly select one of the data lines with RRPV *n*-1, if there is no data line with RRPV *n*-1, RWA will increase the RRPV of all data lines in that set until one data line's RRPV reach *n*-1. Since hot-dirty data lines always have smaller RRPVs, RWA can keep hot-dirty data lines staying in cache to reduce write-backs to PCM.

RWA has the problem that it also keeps single-use dirty lines in cache for a long time, because RWA gives new dirty lines RRPV 0 when they are loaded on cache. This does not help to reduce write-backs to PCM and waste cache capacity. As the improved version,

I-RWA change the way of setting RRPV. For a *n*-way cache, the new line is set to *n*-3 when a read miss, and is set to *n*-2 when a write-back miss. When a data line hits, its RRPV is set to 3 for read hit and is set to 0 for write-back hit. I-RWA policy filters single-use dirty lines , only multiple-used dirty lines have smaller RRPVs are protected from evicting.

Zhou *et al*. [42] proposed write-back-aware cache partitioning (WCP) and write-queue balancing (WQP) replacement policy. WCP and WQP policy are designed for multi-core environment. WCP aims to allocate different cache space size to different cores to reduce write-backs to PCM with maintaining hit ratio. WCP first design monitors to record hit and write-back information for each core. For *M*-way cache, WCP set *M* hit counters and *M* avoidable write-back counters to each core. For each core, its *i*-th hit counter record the number of additional hits and *i*-th avoidable write-back counter record the number of write hits on cache's *i*-th way, when the *i*-th way is available. Based on the information of hit counters and avoidable write-back counters of each core, WCP can make a valid cache space allocation to different cores. The allocation should maximizes the sum of the number of hits and and the weighted number of avoidable write-backs. The weight determines the relative importance of reducing write-backs and raising hit ratio.

WQB replacement policy aims to averagely distribute write-backs among write queues to avoid frequently fulfill one write queue and reduce the delays due to write-backs from LLC. When a dirty line evict from LLC, it enters the corresponding write queue. When a cache miss occurs and no invalid entry can be used, WQP scans from LRU to MRU to find a clean victim or a victim mapped to a light load write queue.

Wang *et al*. [43] present a write-back-aware dynamic cache management technique WADE. WADE predicts the frequently written back blocks in LLC and classifies LLC sets into a frequent write-back list and a non-frequent write-back list, as shown in Figure 2. Since the size of the frequent write-back list affect performance significantly, to get the best performance by balancing reduction of write-backs from LLC to PCM with reduction of cache miss, WADE make an effort to keep proper segment size of frequent write-back list in the LLC. To distinguish frequently written cache blocks, WADE designed a predictor called FWP for frequently written blocks. To find proper size of frequent write-back list for the performance, WADE adopts a segment predictor [44] to dynamically adjust the optimal size of frequent write-back list for each cache set. WADE set different miss penalty for clean cache block and dirty cache block, and the segment predictor can determines the optimal size depending on the miss penalty. When a cache miss occurs, WADE evict a block in non-frequent write-back list if the size of frequent write-back list is not higher than the optimal size, else a block in frequent write-back list is evicted.

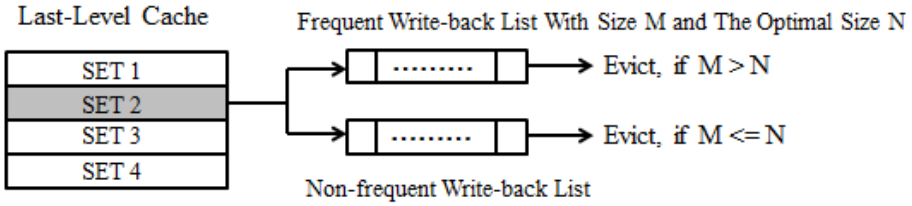

Figure 2. The Overview of Frequent Write-Back List and Non-Frequent Write-Back List Mechanism

Yoo *et al*. [45] present the least-dirty-first (LDF) cache replacement policy. LDF is adopted to on-chip LLC and it reduces PCM write traffic by taking advantages of cache line dirtiness management. The LLC architecture in LDF is shown as Figure 3. For each cache block, LDF set dirty bits to its all cache lines. When a cache line is flushed from L2 cache, its dirty bit is set. The more dirty cache lines a cache block has, the more PCM writes will be incurred when the block is evicted. In order to reduce PCM writes from LLC and maintain cache's hit ratio, LDF evicts the block

with the least number of dirty cache lines among blocks less likely to be referenced again. To find these candidate blocks, LDF maintains one or more reference bits to each cache block, and adopts cache replacement algorithms such as NRU [46] and RRIP. Compared with original NRU and RRIP, NRU and RRIR that be incorporated with LDF reduce PCM writes while maintain cache performance.

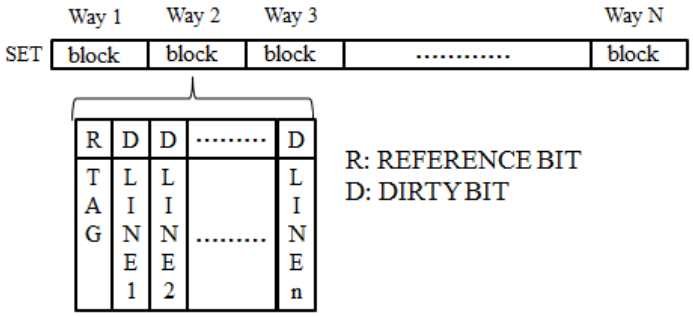

Figure 3. The LLC Set Mechanism in LDF

Rodríguez *et al.* [47] present several new DRRIP-based LLC management policies. DRRIP [26] can be resolved into three sub-policies: insertion sub-policy that initialize the state of new cache blocks, promotion sub-policy that update a cache block's state when it is hit, and victimization sub-policy that select a victim by comparing states of blocks. These policies, including SD, PL, PM, VL and VM, change the different sub-policies of DRRIP to reduce the number of evicting dirty blocks. SD change the insertion sub-policy, it use the number of write-backs instead of the number of cache miss to determine which insertion policy (SRRIP or BRRIP) should be employed to insert a new cache block. PL and PM change the promotion sub-policy by updating RRPV in different ways. VL and VM change the victimization sub-policy by giving a preference to evict clean blocks. The performance evaluation of different policies-combination schemes shows that PM-VH-SD scheme reports the highest reduction of PCM write traffic.

Barcelo *et al*. [48] present two energy efficient LLC replacement policy AL and VL. AL is designed to keep blocks that have been written for more times in cache, based on the observation that the more time a cache block has been written, the more bits in the block have been changed, so the more time and energy is needed to write the block back to PCM when the block is evicted. AL set a Time-To-Live (TTL) value to each blocks in LLC. Each time a block in LLC hits by write request from the upper-level cache, the TTL of the block increases. When a cache miss occurs, AL evicts the least-recently-used block among the blocks with minimum TTL, and decreases the TTL of all blocks to avoid keeping cold blocks in the cache for a long time. So hot-dirty blocks have higher TTL and stay in LLC. VA set an "age" to each blocks in cache, each time a request access LLC, VA increase the ages of all blocks. For a clean page, the age increases by 1; for a dirty page, the age increases by 1/*c*, *c* is according to the average cost of writing a block to PCM. VA evicts the block with the highest age, so clean blocks is preferred to be evicted since their ages increase faster than dirty blocks' ages.

3.2 Last-level Cache Management Based on Hybrid PCM-DRAM Memory

Due to the different characteristics of PCM and DRAM, hybrid memory-based cache management policies of LLC should be aware of cache blocks from DRAM and cache blocks from PCM and manage different types of blocks in different ways.

Weiwei *et al.* [49] proposed a new performance metric TMPKI to measure the performance of LLC for hybrid PCM-DRAM memory, and a hybrid-memory-aware

LLC space management policy called HAP. TMPKI is based on current performance metric MPKI, but it takes into account the different cache miss cost of DRAM data miss and PCM data miss. If the access latency of PCM is L_{ratio} times higher than that of DRAM, TMPKI think that the cost of a PCM data miss is L_{ratio} times higher than that of a DRAM data miss and add a weight to each PCM data miss, so TMPKI present a metric T_{mc} to measure total miss cost. T_{mc} is calculated as follow (D_{mc} *and* N_{mc} means the number of DRAM data misses and PCM data misses):

$$T_{mc} = D_{mc} + N_{mc} \times L_{ratio}$$

HAP manages LLC by properly partition LLC space for DRAM data and PCM data, according to experiment observation: both too many DRAM data misses and too many PCM data miss will decrease performance. HAP set a data type bit to each cache line to identify where the data is from, and two threshold values T_{low} and T_{high} to restrict cache size for PCM. For different workloads, T_{low} and T_{high} are dynamically adjusted according to current TMPKI. HAP adjust cache size for DRAM and PCM by evicting different types of lines when a replacement is triggered: if PCM lines count is higher than T_{high}, HAP evicts a LRU DRAM line; if PCM lines count is lower than T_{low}, HAP evicts a LRU PCM line; else, a LRU line of all lines in cache will be evicted.

4. DRAM Cache Management

DRAM cache of PCM main memory is designed for serve the page accesses, especially write accesses to main memory. To improve the hit ratio, cache management policies of DRAM cache consider prefetching data from PCM. To serve more write accesses and reduce write-backs to PCM, the policies should be able to identify the write-intensive blocks and give them more chance to stay in cache.

Ferreira *et al.* [50] present a clean-preferred victim selection policy (CLP) called *N*-Chance to manage DRAM cache for PCM main memory. *N*-chance policy is based on LRU but it prefers to choose clean cache block to evict. The policy set an parameter *N* to control the preference to select clean pages as victims. When a miss occur in cache, *N*-chance selected the oldest clean cache block as victim among the last *N* least recently used blocks in LRU list, if there is no such block, the LRU block is selected. *N*-chance policy decrease cache's hit ratio, but it reduce write-back to PCM. Due to PCM's high write cost, *N*-chance policy reduce latency and energy by trading write-backs to reads.

Bian *et al.* [51] proposed a flexible DRAM buffer (FDB) for PCM main memory. FDB uses DRAM as an adapting buffer between LLC and PCM main memory to reduce PCM writes and LLC miss penalty. When a dirty block is evicted from LLC, the block is first written back to FDB. Consider that the block in FDB is likely to be fetched to LLC and written again, FDB reduces write-backs to PCM and hides the write latency of PCM. When a cache miss occurs in LLC, while the requested block is transmitted to the LLC, its next one block is prefetched into FDB. Due to the spatial locality, the prefetched block in FDB is likely to be fetched into the LLC later, so FDB hide the read latency of PCM and reduce LLC miss penalty. When FDB is full, it uses FIFO policy to evict a page, and the dirty evicted pages are written back to PCM.

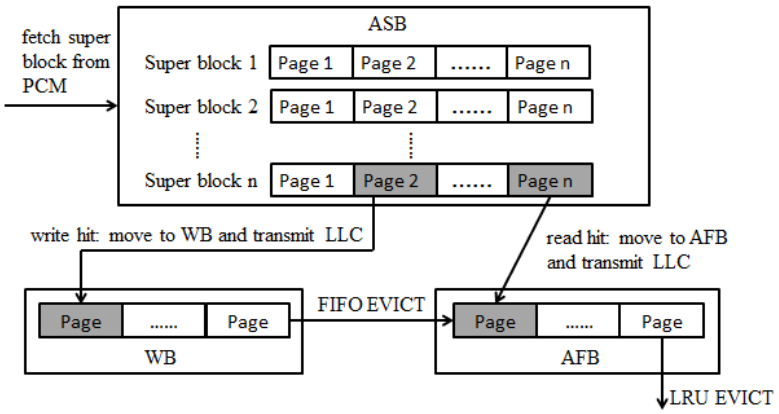

Figure 4. The Overview of DRAM Buffer Consisting of ASB, WB and AFB

Choi *et al.* [52] proposed a DRAM buffer management scheme to minimize PCM main memory accesses to reduce buffer miss rate and PCM writes. As shown in Figure 4, the DRAM buffer is divided into three buffer region: aggressive streaming buffer (ASB), write buffer (WB) and adaptive filtering buffer (AFB). To improve buffer's hit ratio, the DRAM management scheme exploits spatial locality by using ASB to pre-fetch a superblock consisting of a certain number of pages each time, and exploits temporal locality by using AFB to manage the recently referenced pages in the LRU list. To reduce PCM writes, when a page in ASB hits by write request, the page is moved to WB first and evicted to AFB later, so the dirty pages are delayed to be written into PCM. The DRAM buffer scheme works as follow: when a page request from LLC hit the DRAM buffer, if the requested page is in ASB, the page is moved to AFB for read request or WB for write request; If WB has no free space, FIFO replacement policy is used to evict a page to AFB; If AFB has no free space, the LRU page is evicted to PCM; when a miss occurs in the DRAM buffer, a superblock containing the requested data is pre-fetched into ASB, if ASB has no free space, the victim is selected with FIFO.

Jang *et al.* [53] proposed a data classification management scheme to manage DRAM buffer for hybrid SLC/MLC PCM main memory. As shown in Figure 5, the DRAM buffer is divided into two buffer regions: aggressive fetching superblock buffer (AFSB) and selective filtering buffer (SFB). The AFSB caches the superblocks from PCM, the size of a superblock is the same as the page unit managed in main memory and storage by OS. Each superblock in AFSB is partitioned into a certain number of sub-pages, the size of a sub-page is the same as the cache block of LLC. Each sub-page's access count is recorded in AFSB, and the average access count of all sub-pages of a superblock is set as the threshold to classify the superblock's high-accessed sub-pages and low-accessed sub-pages. To maintain the hit ratio of DRAM buffer, when a superblock is evicted from AFSB, SFB caches the superblock's all high-accessed sub-pages to give these sub-pages a second chance. The DRAM buffer scheme works as follow: when a sub-page is requested from LLC, AFSB and SFB are checked for the requested sub-page, if the sub-page exists in AFSB or SFB, the sub-page is fetched into LLC; when a miss occurs in both AFSB and SFB, the superblock that contains the requested sub-page is fetched into AFSB, if AFSB has no free space, a superblock with the lowest average access count of its all sub-pages is evicted, and the victim superblock's high-accessed sub-pages are fetched into SFB.

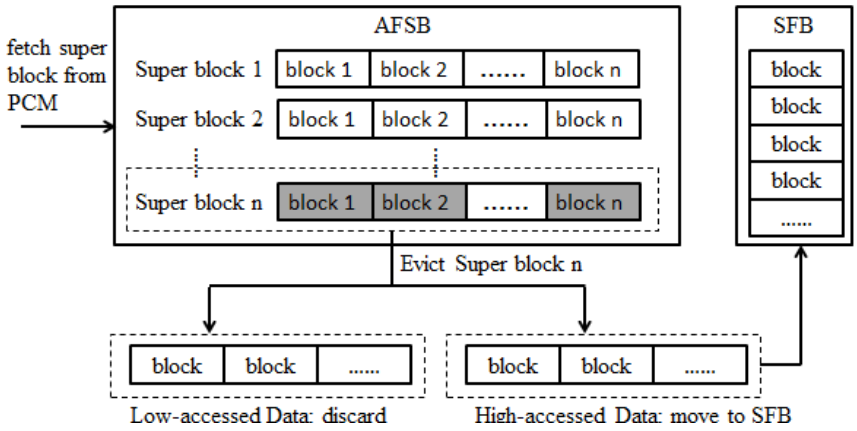

Figure 5. The Overview of DRAM Buffer Consisting of AFSB and SFB

Park *et al.* [54] proposed a weal-level replacement algorithm to manage DRAM buffer for PCM main memory. The algorithm considers both reducing PCM write count and uniform PCM write count distribution. The DRAM buffer is divided into two layers: in the first layer, blocks are managed by LRU list with both clean blocks and dirty blocks; in the second layer, dirty blocks and clean blocks are managed separately, dirty blocks are maintained in the write frequency list according to their corresponding PCM blocks' write count, and clean blocks are managed by LRU list. When the DRAM buffer is full, to reduce write-backs to PCM, the algorithm first evicts the least recently used clean block in second layer; if no clean block exists in the second layer, to even out PCM write count, the dirty block with the least write count of its corresponding PCM block is evicted; if the dirty blocks have the same write count, the least recently used dirty block is evicted.

5. Main Memory Management

5.1 Pure PCM Main Memory Management

Each page cell of PCM allows limited write operation count. To prevent too many write accesses focus on the same PCM location due to page placement and replacement, cache management policies of PCM main memory should care about the physical address when a page is placed or replaced on PCM.

Yi Ou *et al.* [55] proposed several wear-aware page replacement algorithms including LPW, LFM and LRM for PCM-based database buffer pools. The algorithms are designed for PCM wear leveling. LFM and LPM monitors PCM pages' wear status at the page-level. When page fault occurs and a victim is needed, LFM selects the least frequently modified (including page update and page replacement) PCM page, while LRM selects the least recently modified PCM page. LPW consider that the different parts inside a PCM page have different wear status, monitoring wear status at the page-level (number of writes on each page) is not sufficient. LPW splits pages in PCM into a number of wear units. The page wear of a PCM page is defined as the total write count of all wear units in the page. When a page updated, the page wear increases by the number of updated wear units. When a page replacement occurs, the page wear of the buffer page where the victim resided increases by the number of wear units in the page. For PCM wear-leveling, when page fault occurs, LPW evicts the page having the smallest page wear count. Since the algorithms achieve PCM wear leveling, they lower the hit ratio of PCM buffer compared to LRU algorithm.

5.2 Hybrid PCM-DRAM Memory Management

To improve the overall performance of hybrid memory, the ideal situation is that DRAM can serve as more write accesses as possible. Cache management policies of hybrid memory pay attention to predict pages' access pattern and classify pages into write-hot page and read-hot page (or write-cold page). By performing page allocation and page migration between PCM and DRAM, the policies keep write-hot pages in DRAM and read-hot pages (or write-cold pages) in PCM to serve more write accesses on DRAM.

Seok *et al.* [56] proposed an LRU-based page management algorithm for hybrid main memory (thereinafter the algorithm is called MIG). MIG adopts a prediction mechanism to identify read-intensive pages and write-intensive pages according to pages' access pattern, and a page migration mechanism to migrate read-intensive pages to PCM and write-intensive pages to DRAM. The prediction mechanism set a weight value to each page, the weight value increases when the page hits by read request, and decreases when the page hits by write request. For pages in PCM and DRAM, the algorithm classified them into read queue and write queue according to the weight value, and all pages in hybrid memory are queued in the LRU list. When a page fault occurs and page eviction is needed, the algorithm evicts the LRU page in the LRU list. When a page hits, the algorithm calculate the page's new weight value according to the type of the request, then judge if the page should be moved to the other queue in the same memory or migrated to the other memory according to the two thresholds *Trq* and *Trmig* that are set by the algorithm. When a page migration occurs, the migration target memory should have enough space for the migrated page. If DRAM does not have enough space to place the migrated page, DRAM evicts the LRU page of its read queue. If PCM is full, PCM evicts the LRU page of its write queue for the migrated page. Figure 6 shows an example of page migration in MIG. Since MIG can reduce PCM writes, it has some problems that affect performance: first, MIG evicts pages not only when page fault occurs, but also when page migration occurs, this extra page releases will introduce more page faults and degrade hit ratio, consider that the latency of second storage such as disk and SSD is order of magnitude higher than PCM write latency, sacrificing hit ratio to reduce PCM writes may lower the overall performance; Second, MIG migrates read-intensive pages from DRAM to PCM, that may cause more write traffic on PCM due to page placement and migration for read-tendency workloads.

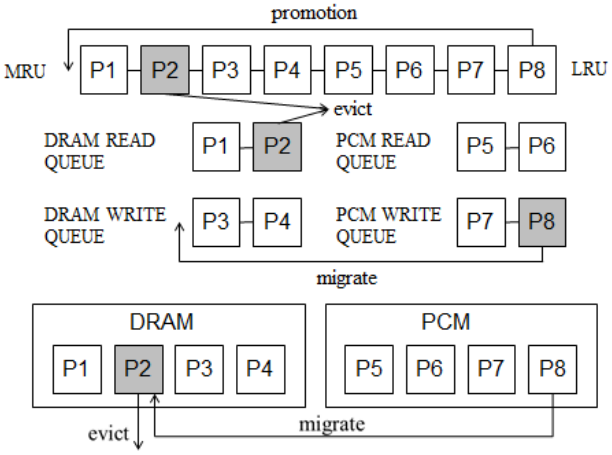

Figure 6. An Example of Page Migration in MIG

Lee *et al.* [57] present a CLOCK-based write-history-aware page replacement algorithm CLOCK-DWF. CLOCK-DWF manages DRAM pages and PCM pages in separate circular lists. Each page in hybrid memory has a reference bit and a dirty bit. Dirty bit is used to identify the frequently written pages in DRAM, a DRAM page's dirty bit is set to 1 when the page hits by a write request. CLOCK-DWF keeps write-intensive pages in DRAM as follow: when a page fault occurs, CLOCK-DWF put the requested page in DRAM if the request is a write request, otherwise the page is put on PCM; when a page in PCM hits by a write request, the page should be migrated to DRAM immediately. To get a free frame on DRAM when DRAM is full, CLOCK-DWF scan the DRAM circular list to selects a write-cold page in DRAM and migrate the page to PCM. To get a free frame on PCM when PCM is full, traditional CLOCK algorithm is used on PCM circular list to evict a non-recently used page. Figure 7 shows an example of page migration when a PCM page hit by write request. CLOCK-DWF also have some problems: First, CLOCK-DWF does not always evict the least recently used page in hybrid memory, compared with original CLOCK or LRU, CLOCK-DWF may have a higher page fault count; Second, CLOCK-DWF does not consider whether DRAM is full or not when it performs page migration from PCM to DRAM, migrating a PCM page to a full DRAM causes migrating a DRAM page back to PCM, this page exchange is unnecessary if the PCM page is only written one time, and may cause more page exchange between DRAM and PCM in the future if the page migrated to PCM hits by write request; Third, CLOCK-DWF places faulted pages only according to the type of requests, it may causes more write traffic on PCM in read-intensive workloads due to page placement and replacement, and it may waste hybrid memory capacity and consequently introduce more page faults, an extreme example is that for a read-only application such as search engine, only PCM capacity is used.

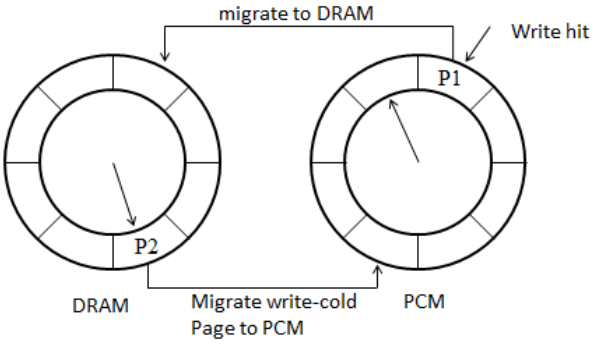

Figure 7. An Example of Page Migration In MIG

Wu *et al.* [58] proposed a page replacement algorithm APP-LRU. Unlike MIG and CLOCK-DWF, APP-LRU aims to reduce PCM writes without degrading the hit ratio of hybrid memory. APP-LRU maintains three lists: a LRU list to manage all pages in hybrid memory, a DRAM-list to queue DRAM pages in order of read count and a PCM-list to queue PCM pages in order of write count. To reduce PCM writes, APP-LRU determines putting the requested page on DRAM or PCM according to the page's history accesses in memory when page fault occurs. To achieve this goal, APP-LRU maintains a metadata table to record a page's history accesses when the page is evicted. When a page request misses, APP-LRU selects the LRU page in hybrid memory as victim. If the location of the victim is not fit for the requested pages, for DRAM (PCM) victim, APP-LRU frees the victim and migrates the head page of PCM-list (DRAM-list) is migrated to the free frame, then put the requested page on the befitting memory. Figure 8 gives an example of APP-LRU page

replacement for a requested page that should be put on DRAM. APP-LRU has the same hit ratio as original LRU since APP-LRU evicts the LRU page and does not release extra pages. But APP-LRU only perform page migration during page replacement, that will cause two problems: first, for large capacity memory with little page faults, APP-LRU takes little effect, even no effect if no page fault occurs; Second, if a PCM pages become write-intensive and hits by multiple write requests, APP-LRU doesn't works to reduce PCM writes for this situation.

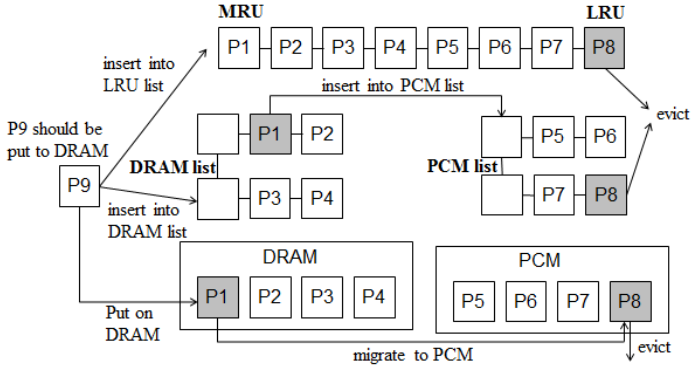

Figure 8. An Example of APP-LRU Page Replacement

Chen *et al.* [59] proposed a page replacement algorithm MHR-LRU. MHR-LRU considers that the most recently written pages have high possibility to be written again, to reduce PCM writes, when page fault occurs, the requested page should be put on DRAM if the request type is write. MHR-LRU queues all pages in a LRU list, and queues all DRAM pages in orders of their most recent write reference time. When a page request misses, the page in LRU position of LRU list is selected as victim. For a write request, if the victim is on PCM, instead of putting the requested page on PCM, MHR-LRU moves the least recently written page in DRAM to PCM and put the requested page on DRAM, as shown in Figure 9. Like APP-LRU, MHR-LRU has the same hit ratio as LRU, but takes little or no effect if the hit ratio is very high.

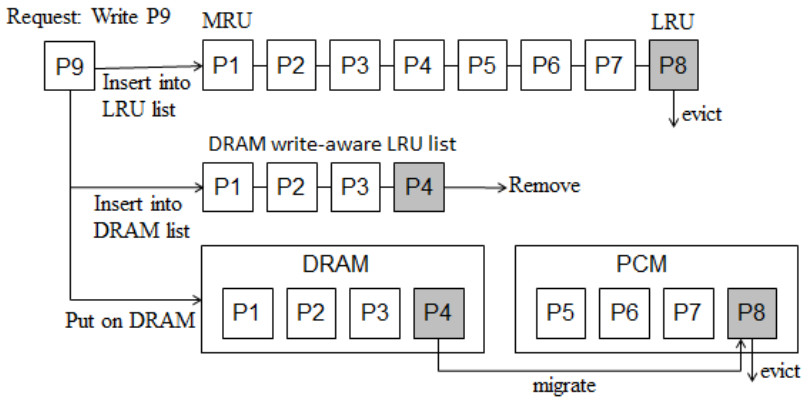

Figure 9. MHR-LRU Page Migration to Put the Requested Page on DRAM

Heish *et al.* [60] proposed a double circular page management scheme DCCS. DCCS maintains DRAM pages and PCM pages separately in a DRAM circular LRU list and a PCM circular LRU list. DCCS set an operation bit to each page in DRAM and PCM, the operation bit is set to record the last request type performed on the page. DCCS identifies read-bound pages according to their operation bits. To avoid frequent writes to PCM due to page replacement, DCCS only use DRAM to serve

page faults. When page fault occurs, the requested page is put on DRAM. When a page in PCM hits by write request, DCCS migrates the page to DRAM. If DRAM has no enough space, DCCS evicts the LRU page in DRAM and moves the most recently used read-bound page to PCM. If PCM has no enough space, DCCS evicts the LRU page in PCM. Figure 10 shows how DCCS get free pages on DRAM. DCCS also have some problems: first, DCCS evict the LRU page in DRAM, not the LRU page in hybrid memory, that affect the hit ratio; second, when DRAM and PCM are full, a page fault will make both the LRU page in DRAM and the LRU page in PCM evicted, that introduces more page faults; third, DCCS determines read-bound pages only depend on their last request type, the metric is not accurate to reflect page's access pattern.

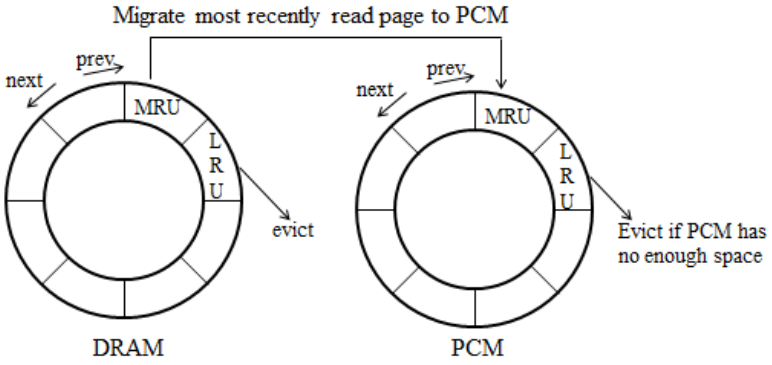

Figure 10. DCCS Gets Free Pages on DRAM

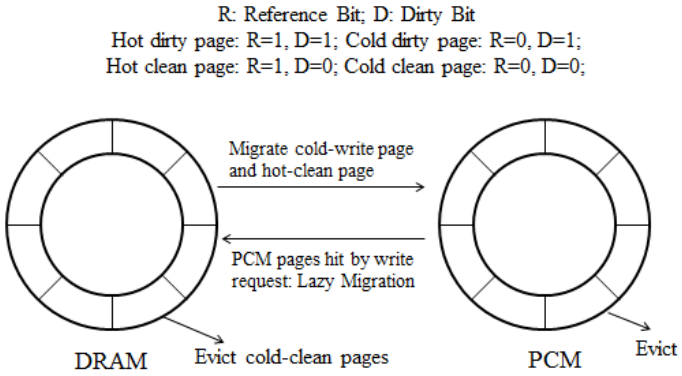

Figure 11. The Overview of M-CLOCK

Lee *et al.* [61] proposed a migration-optimized page replacement algorithm M-CLOCK. M-CLOCK aims to reduce PCM writes while have a high hit ratio and reduce unnecessary migration that causes migration-thrashing. The overview of M-CLOCK is shown in Figure 11. To reduce PCM writes, M-CLOCK place all faulted pages on DRAM and keep write-intensive pages in DRAM. To classify write-intensive pages, M-CLOCK set a reference bit and a dirty bit to each page, when a dirty page in DRAM with the reference bit 1 is re-referenced again, the page is considered as a write-intensive page. M-CLOCK manages write-intensive pages in DRAM by a clock-hand called D-hand and manages other pages in DRAM by a clock-hand called C-hand. When the DRAM is full, to get a free page, M-CLOCK first uses D-hand to scan write-intensive pages to degrade a write-intensive page into cold dirty page, then uses C-hand to search for a hot-clean page or a cold-dirty pages in DRAM to be migrated to PCM, or a cold clean pages in DRAM to be evicted. To reduce unnecessary page migrations, M-CLOCK uses a lazy migration policy depending on setting a lazy bit to each page in PCM. When a page in PCM

hits by write request, if no free page exists in DRAM and the page's lazy bit is unset, the page is updated in-place and its lazy bit is set; otherwise, the page should be migrated to DRAM.

6. DRAM Memory Management Based on PCM Storage

Unlike disk and flash memory SSD, there is not huge access latency gap between PCM storage and DRAM. Since the cache miss cost is not unacceptable, and caching pages from PCM into DRAM also cause the system overhead, so traditional policies that cache all referenced pages from storage into memory need to be rethought. Besides, to reduce write-backs from DRAM to PCM, the page replacement policy of DRAM should also be rethought.

Yoo *et al.* [62] proposed a CLOCK-based page replacement algorithm LDF-CLOCK (least-dirty-first CLOCK) to manage DRAM main memory for PCM swap device. LDF-CLOCK considers both recency and dirtiness to evict a page when page replacement is needed. Like CLOCK, LDF-CLOCK also manages pages in DRAM in the circular list and set one reference bit per page. To mark the dirtiness level of a page, LDF-CLOCK partition each page into several sub-pages, and set one dirty bit per sub-page. When a write access comes, LDF-CLOCK sets the dirty bits of the written sub-pages in the accessed page to 1. A page with more dirty sub-pages makes more write-backs to PCM when it is evicted. To select a victim, as shown in Figure 12, LDF-CLOCK uses the clock-hand to scan each page in the circular list, if the page being scanned has the reference bit of 1, the reference bit is set to 0 and LDF-CLOCK continue to scan the next page until the clock-hand points to a page with the reference bit of 0. After the scan, LDF-CLOCK selects the page with least dirty sub-pages among all pages with reference bit of 0. To overcome the time complexity of finding the least dirty page, LDF-CLOCK maintains a multiple lists structure to keep pages with reference bit of 0 in different lists according to the number of their dirty sub-pages.

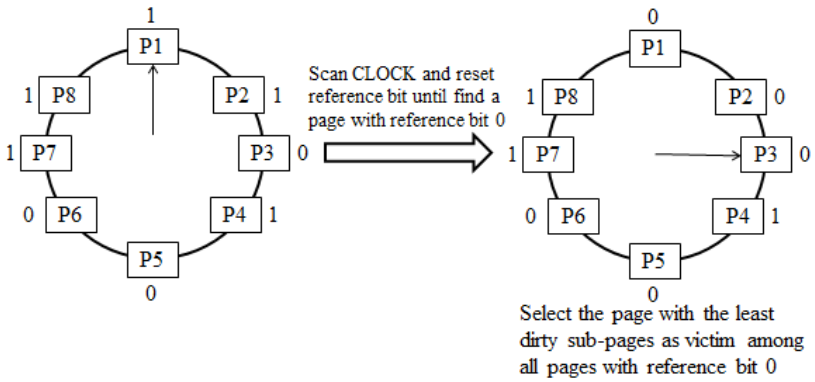

Figure 12. Select the Victim in LDF-CLOCK

Lee *et al.* [63] proposed a cache management scheme called selective cache bypassing (thereinafter it is abbreviated to SCB) to manage DRAM main memory for PCM-based storage. SCB rethinks DRAM cache strategy that caching PCM data selectively in DRAM is more efficient than caching all requested data in DRAM. Since PCM access latency is slightly slower than DRAM, compared with the cost of a cache miss, the overhead of caching a page in DRAM should be consider. If a requested block is caching into DRAM but it is not referenced again before it is evicted, SCB thinks caching this block has adverse effect on performance, because it costs more than accessing the page directly on PCM and wastes cache space, so

this single-accessed block should not be cached. Based on this observation, to filter out the single-accessed blocks from caching, SCB does not cache a block for its first access, the block is caching into DRAM after it is accessed again within a certain number of distance, the default distance is the same as the cache size. The reason to bypass the first accesses of blocks is that most of blocks are accessed either 0 or many times, so the second access within a short time can be an indicator for multiple-accessed blocks. The performance profit of SCB is determined by the ratio of single-access blocks of workloads, so SCB adopt adaptive strategy that monitors the dynamic situations of the workload and adaptively uses R-bypass policy (bypass read accesses), RW-bypass policy (bypass both read accesses and write accesses), or NO-bypass.

7. Conclusion

PCM is flexible to be adopted as either main memory or storage, but its advantages and drawbacks are different from both DRAM and disk, so PCM systems require special cache management schemes. In this paper, we survey PCM-aware cache management policies to manage LLC and main memory for different types of PCM systems. We discuss the new challenges for cache management in different PCM system architectures. Then we introduce and analyze the novel cache management policies at different memory levels for PCM main memory, hybrid PCM-DRAM memory and PCM storage.

Although there are no widely-accepted solutions to using PCM in current memory hierarchy, a general idea for improving PCM-based systems is to reduce writes to PCM. Using appropriate caches is considered as an efficient approach for reducing PCM writes. Since the low read access latency of PCM can significantly reduce the cost of cache misses, PCM-aware cache management policy needs to identify the blocks that are high likely to be written instead of the blocks that are high likely to be re-referenced. Thus, a future research direction is to study the prediction on future write-request trend.

Acknowledgement

This work is partially supported by the National Science Foundation of China under the grant (61379037, 61472376) and the Fundamental Research Funds for the Central Universities. Peiquan Jin is the corresponding author.

References

[1] G. W. Burr and B. N. Kurdi, “Overview of candidate device technologies for storage-class memory”. IBM Journal of Research and Development, vol. 52, no. 4-5, **(2008)**, pp. 449-464.
[2] S. Raoux, G. W. Burr and M. J. Breitwisch, “Phase-change random access memory: A scalable technology”, IBM Journal of Research and Development, vol. 52, no. 4-5, **(2008)**, pp. 465–479.
[3] B. C. Lee, E. Ipek and O. Mutlu, “Architecting phase change memory as a scalable dram alternative”, Proceedings of the 36th International Symposium on Computer Architecture, Austin, Texas, USA, June 20-24, **(2009)**.
[4] M. K. Qureshi, V. Srinivasan and J. A. Rivers, “Scalable high performance main memory system using phase-change memory technology”, Proceedings of the 36th International Symposium on Computer Architecture, Austin, Texas, USA, June 20-24, **(2009)**.
[5] P. Zhou, B. Zhao, J. Yang and Y. Zhang, “A durable and energy efficient main memory using phase change memory technology”, Proceedings of the 36th International Symposium on Computer Architecture, Austin, Texas, USA, June 20-24, **(2009)**.
[6] B. C. Lee, P. Zhou, J. Yang and Y. Zhang, “Phase-change technology and the future of main memory”, IEEE Micro, vol. 30, no. 1, **(2010)**, pp. 131-141.
[7] H. Yoon, N. Muralimanohar and J. Meza, “Data mapping and buffering in multi-level cell memory for higher performance and energy efficiency”, CMU SAFARI Tech. Report, **(2013)**.

[8] H. Yoon, J. Meza and N. Muralimanohar, "Efficient data mapping and buffering techniques for multilevel cell phase-change memories", ACM Transactions on Architecture and Code Optimization, vol. 11, no. 4, **(2014)**, pp. 40:1-25.
[9] M. K. Qureshi, M. M. Franceschini and L. A. Lastras-Montano, "Improving read performance of phase change memories via write cancellation and write pausing", Proceedings of the 16th International Conference on High-Performance Computer Architecture, Bangalore, India, January 9-14, **(2010)**.
[10] H. G. Lee, S. Baek and C. Nicopoulos, "An energy- and performance-aware DRAM cache architecture for hybrid DRAM/PCM main memory systems", Proceedings of the 29th IEEE International Conference on Computer Design, Amherst, Massachusetts, USA, October 9-12, **(2011)**.
[11] T. J. Ham, B. K. Chelepalli and N. Xue, "Disintegrated control for energy-efficient and heterogeneous memory systems", Proceedings of the 19th IEEE International Symposium on High Performance Computer Architecture, Shenzhen, China, February 23-27, **(2013)**.
[12] G. Dhiman, R. Ayoub and T. Rosing, "PDRAM: A hybrid PRAM and DRAM main memory system", Proceedings of the 46th Design Automation Conference, San Francisco, California, USA, July 26-31, **(2009)**.
[13] L. E. Ramos, E. Gorbatov and R. Bianchini, "Page placement in hybrid memory systems", Proceedings of the 25th International Conference on Supercomputing, Tucson, Arizona, USA, May 31 - June 04, **(2011)**.
[14] H. B. Yoon, J. Meza and R. Ausavarungnirun, "Row buffer locality aware caching policies for hybrid memories", Proceedings of the 30th International IEEE Conference on Computer Design, Montreal, QC, Canada, September 30–October 3, **(2012)**.
[15] E. Lee, J. Jang and H. Bahn, "DTFS: Exploiting the similarity of data versions to design a write-efficient file system in phase-change memory", Proceedings of the 29th ACM Symposium on Applied Computing, Gyeongju, Korea, March 24-28, **(2014)**.
[16] E. Lee, J. Jang and T. Kim, "On-demand snapshot: an efficient versioning file system for phase-change memory", IEEE Transactions on Knowledge and Data Engineering, vol. 25, no. 12, **(2013)**, pp. 2841-2853.
[17] E. Lee, S. Yoo and J. E. Jang, "Shortcut-JFS: a write efficient journaling file system for phase change memory", Proceedings of the IEEE 28th IEEE Symposium on Mass Storage Systems and Technologies, Piscataway, NJ, USA, April 16-20, **(2012)**.
[18] G. S. Choi, B. W. On and K. Choi, "PTL: PRAM translation layer", Microprocessors and Microsystems, vol. 37, pp. 24-32, **(2012)**.
[19] S. Im and D. Shin, "Differentiated space allocation for wear leveling on phase-change memory-based storage device", IEEE Transactions on Consumer Electronics, vol. 60, no. 1, **(2014)**, pp. 45-51.
[20] A. Dan and D. Towsley, "An Approximate Analysis of the LRU and FIFO Buffer Replacement Schemes", Proceedings of the 1990 ACM SIGMETRICS conference on Measurement and modeling of computer systems, Boulder, Colorado, USA, May 22-25, **(1990)**.
[21] E. J. O'Neil, P. E. O'Neil and G. Weikum, "The LRU-K Page Replacement Algorithm for Database Disk Buffering", Proceedings of the ACM SIGMOD International Conference on Management of Data, Washington, D.C., USA, May 26-28, **(1993)**.
[22] Y. Zhou and J. F. Philbin, "The Multi-Queue Replacement Algorithm for Second Level Buffer Caches", Proceedings of the USENIX Annual Technical Conference, Boston, Massachusetts, USA, June 25-30, **(2001)**.
[23] T. Johnson and D. Shasha, "2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm", Proceedings of 20th International Conference on Very Large Data Bases (VLDB), Santiago de Chile, Chile, September 12-15, **(1994)**.
[24] S. Jiang and X. Zhang, "LIRS: An efficient low inter-reference recency set replacement policy to improve buffer cache performance", Proceedings of the International Conference on Measurements and Modeling of Computer Systems (SIGMETRICS), Marina Del Rey, California, USA, June 15-19, **(2002)**.
[25] C. Kim, J. Choi and J. Kim, "LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies", IEEE Transactions on Computers, vol. 50, no. 12, **(2001)**, pp. 1352–1361.
[26] A. Jaleel, K. Theobald and S. Steely Jr., "High performance cache replacement using re-reference interval prediction (RRIP)", Proceedings of the 37th International Symposium on Computer Architecture, Saint-Malo, France, June 19-23, **(2010)**.
[27] M. Chaudhuri, "Pseudo-LIFO the foundation of a new family of replacement policies for last-level caches", Proceedings of the 42st Annual IEEE/ACM International Symposium on Microarchitecture, New York, New York, USA, December 12-16, **(2009)**.
[28] S. Mittal, "Energy saving techniques for phase change memory (PCM)", arXiv: 1309.3785, **(2013)**.
[29] M. K. Qureshi, S. Gurumurthi and B. Rajendran, "Phase Change Memory: From Devices to System", Synthesis Lectures on Computer Architecture, vol. 6, no. 4, **(2011)**, pp. 1-134.
[30] O. Zilberberg, S. Weiss and S. Toledo, "Phase-change memory: An architectural perspective", ACM Computing Survey, vol. 45, no. 3, **(2013)**, pp. 1–33.

[31] S. Cho and H. Lee, "Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance", Proceedings of the 42st Annual IEEE/ACM International Symposium on Microarchitecture, New York, New York, USA, December 12-16, **(2009)**.
[32] B. D. Yang, J. E. Lee and J. S. Kim, "A Low Power Phase-Change Random Access Memory using a Data-Comparison Write Scheme", Proceedings of the 2007 IEEE International Symposium on Circuits and Systems, Piscataway, NJ, USA, May 20-27, **(2007)**.
[33] J. Yue and Y. Zhu, "Accelerating write by exploiting PCM asymmetries", Proceedings of the 19th IEEE International Symposium on High Performance Computer Architecture, Shenzhen, China, February 23-27, **(2013)**.
[34] Y. Du, M. Zhou and B. R. Childers, "Bit mapping for balanced PCM cell programming", Proceedings of the 40th Annual International Symposium on Computer Architecture, Tel-Aviv, Israel, June 23-27, **(2013)**.
[35] M. K. Qureshi, M. M. Franceschini and A. Jagmohan, "PreSET: Improving performance of phase change memories by exploiting asymmetry in write times", Proceedings of the 39th International Symposium on Computer Architecture, Portland, OR, USA, June 9-13, **(2014)**.
[36] F. Xia, D. Jiang and J. Xiong, "DWC: Dynamic write consolidation for phase change memory systems", Proceedings of the 2014 International Conference on Supercomputing, Muenchen, Germany, June 10-13, **(2014)**.
[37] A. Jaleel, W. Hasenplaugh and M. Qureshi, "Adaptive insertion policies for managing shared caches", Proceedings of the International Conference on Parallel Architectures & Compilation Techniques, Toronto, Ontario, Canada, October 25-29, **(2008)**.
[38] M. K. Qureshi, A. Jaleel and Y. N. Patt, "Adaptive insertion policies for high performance caching", Proceedings of International Symposium on Computer Architecture, San Diego, California, USA, June 9-13, **(2007)**.
[39] R. Freitas and W. Wilcke, "Storage-class memory: The next storage system technology", IBM Journal of Research and Development, vol. 52, no. 4, **(2008)**, pp. 439-447.
[40] Ru Fang, Hui-I Hsiao and Bin He, "High performance database logging using storage class memory", Proceedings of the 27th International Conference on Data Engineering, Hannover, Germany, April 11-16, **(2011)**.
[41] Zhang Xi, Qian Hu and Dongsheng Wang, "Read-write aware replacement policy for phase change memory", Proceedings of the 9th international conference on Advanced parallel processing technologies, Shanghai, China, September 26-27, **(2011)**.
[42] M. Zhou, Y. Du and B. Childers, "Write-back-aware partitioning and replacement for last-level caches in phase change main memory systems", ACM Transactions on Architecture & Code Optimization, vol. 8, no. 4, **(2012)**, pp. 73-94.
[43] Z. Wang, S. Shan and T. Cao, "WADE: Write-back-aware dynamic cache management for NVM-based main memory system", ACM Transactions on Architecture & Code Optimization, vol. 10, no. 4, **(2013)**, pp. 51:1-51:21.
[44] S. M. Khan, Z. Wang and D. A. Jimenez, "Decoupled dynamic cache segmentation", Proceedings of the 18th IEEE International Symposium on High-Performance Computer Architecture, Washington, DC, USA, February 25-29, **(2012)**.
[45] S. Yoo, E. Lee and H. Bahn, "LDF (less dirty first): dirtiness-aware cache replacement policy for PCM main memory", Electronics Letters, vol. 49, no. 25, **(2013)**, pp. 1607-1609.
[46] NRU, "Inside the Intel Itanium 2 Processor", HP Technical White Paper, **(2002)**.
[47] R. Rodriguez-Rodriguez, F. Castro and D. Chaver, "Reducing writes in phase-change memory environments by using efficient cache replacement policies", Proceedings of the Design, Automation and Test in Europe, Grenoble, France, March 18-22, **(2013)**.
[48] N. Barcelo, M. Zhou and D. Cole, "Energy efficient caching for phase-change memory", Proceedings of the Design and Analysis of Algorithms - First Mediterranean Conference on Algorithms, Kibbutz Ein Gedi, Israel, December 3-5, **(2012)**.
[49] W. Wei, D. Jiang and J. Xiong, "HAP: Hybrid-memory-Aware Partition in shared Last-Level Cache", Proceedings of the 32nd IEEE International Conference on Computer Design, Seoul, Korea, October 19-22, **(2014)**.
[50] A. P. Ferreira, M. Zhou and S. Bock, "Increasing PCM main memory lifetime", Proceedings of the Design, Proceedings of the Automation and Test in Europe, Dresden, Germany, March 8-12, **(2010)**.
[51] M. Y. Bian, S. K. Yoon and S. D. Kim, "An Effective Interfacing Adapter for PRAM Based Main Memory via Flexible Management DRAM Buffer", Proceedings of the International Conference on IT Convergence and Security, Pyeong Chang, Korea, December 5-7, **(2012)**.
[52] I. S. Choi, S. I. Jang and C. H. Oh, "A dynamic adaptive converter and management for PRAM-based main memory", Microprocessors & Microsystems, vol. 37, no. 67, **(2013)**, pp. 554–561.
[53] S. I. Jang, C. G. Kim and S. D. Kim, "An Efficient DRAM Converter for Non-Volatile Based Main Memory", Proceedings of the International Conference on IT Convergence and Security, Pyeong Chang, Korea, December 5-7, **(2012)**.

[54] S. K. Park, M. K. Maeng and K. W. Park, "Adaptive wear-leveling algorithm for PRAM main memory with a DRAM buffer", Acm Transactions on Embedded Computing Systems, vol. 13, no. 4, **(2014)**, pp. 88:1-88:25.
[55] Y. Ou, L. Chen and J. Xu, "Wear-Aware Algorithms for PCM-Based Database Buffer Pools", Proceedings of the Web-Age Information Management International Workshops: BigEM, HardBD, DaNoS, HRSUNE, BIDASYS, Macau, China, June 16-18, **(2014)**.
[56] H. Seok, Y. Park and K. W. Park, "Efficient page caching algorithm with prediction and migration for a hybrid main memory", Acm Sigapp Applied Computing Review, vol. 11, no. 4, **(2011)**, pp. 38-48.
[57] S. Lee, H. Bahn and S. H. Noh, "CLOCK-DWF: A Write-History-Aware Page Replacement Algorithm for Hybrid PCM and DRAM Memory Architectures", IEEE Transactions on Computers, vol. 63, no. 9, **(2014)**, pp. 2187-2200.
[58] Z. Wu, P. Jin and C. Yang, "APP-LRU: A New Page Replacement Method for PCM/DRAM-Based Hybrid Memory Systems", Proceedings of the Network and Parallel Computing - 11th IFIP WG 10.3 International Conference, Ilan, Taiwan, September 18-20, **(2014)**.
[59] K. Chen, P. Jin and L. Yue, "A Novel Page Replacement Algorithm for the Hybrid Memory Architecture Involving PCM and DRAM", Proceedings of the Network and Parallel Computing - 11th IFIP WG 10.3 International Conference, Ilan, Taiwan, September 18-20, **(2014)**.
[60] J. W. Hsieh and Y. H. Kuan, "Double Circular Caching Scheme for DRAM/PRAM Hybrid Cache", Proceedings of the 19th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, Seoul, Korea, August 19-22, **(2012)**.
[61] M. Lee, D. H. Kang and J. Kim, "M-CLOCK: migration-optimized page replacement algorithm for hybrid DRAM and PCM memory architecture", Proceedings of the 30th Annual ACM Symposium on Applied Computing, Salamanca, Spain, April 13-17, **(2015)**.
[62] S. Yoo, E. Lee and H. Bahn, "LDF-CLOCK: The Least-Dirty-First CLOCK Replacement Policy for PCM-based Swap Devices", Journal of Semiconductor Technology & Science, vol. 15, **(2015)**, pp. 68-76.
[63] S. Park, D. Jung and J. Kang, "CFLRU: a replacement algorithm for flash memory", Proceedings of the 2006 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, Seoul, Korea, October 22-25, **(2006)**.
[64] H. Jung, H. Shim and S. Park, "LRU-WSR: integration of LRU and writes sequence reordering for flash memory", IEEE Transactions on Consumer Electronics, vol. 54, no. 3, **(2008)**, pp. 1215-1223.
[65] Y. Lv, X. Chen and B. Cui, "ACAR: An Adaptive Cost Aware Cache Replacement Approach for Flash Memory", Proceedings of the 11th International Conference on Web-Age Information Management, Jiuzhaigou, China, July 15-17, **(2010)**.
[66] P. Jin, Y. Ou and T. Haerder, "ADLRU: An Efficient Buffer Replacement Algorithm for Flash-based Databases", Data and Knowledge Engineering (DKE), vol. 72, **(2012)**, pp. 83-102.
[67] Z. Li, P. Jin and X. Su, "CCF-LRU: A New Buffer Replacement Algorithm for Flash Memory", IEEE Transaction on Consumer Electronics, vol. 55, no. 3, **(2009)**, pp. 1351-1359.
[68] Y. Ou, T. Haerder and P. Jin, "CFDC—A Flash-aware Replacement Policy for Database Buffer Management", Proceedings of the 5th International Workshop on Data Management on New Hardware (DaMoN'09) (in conjunction with SIGMOD'09), Providence, Rhode Island, USA, June 29–July 2, **(2009)**.
[69] I. Koltsidas and S. Viglas, "Flashing up the storage layer", Proceedings of the VLDB Endowment, vol. 1, no. 1, **(2008)**, pp. 514-525.
[70] G. Soundararajan, V. Prabhakaran and M. Balakrishnan, "Extending SSD Lifetimes with Disk-Based Write Caches", Proceedings of the 8th USENIX Conference on File and Storage Technologies, San Jose, CA, USA, February 23-26, **(2010)**.
[71] P. Yang, P. Jin and L. Yue, "Hybrid Storage with Disk Based Write Cache", Proceedings of the First International Workshop on Flash-Based Database Systems (FlashDB) (in conjunction with DASFAA'11), Hong Kong, China, April 22-25, **(2011)**.

