M-CLOCK: Migration-optimized Page Replacement Algorithm for Hybrid Memory Architecture

MINHO LEE, DONG HYUN KANG, and YOUNG IK EOM, Sungkyunkwan University

Phase Change Memory (PCM) has drawn great attention as a main memory due to its attractive characteristics such as non-volatility, byte-addressability, and in-place update. However, since the capacity of PCM is not fully mature yet, *hybrid memory architecture* that consists of DRAM and PCM has been suggested as a main memory. In addition, page replacement algorithm based on *hybrid memory architecture* is actively being studied, because existing page replacement algorithms cannot be used on *hybrid memory architecture* in that they do not consider the two weaknesses of PCM: high write latency and low endurance. In this article, to mitigate the above hardware limitations of PCM, we revisit the page cache layer for the *hybrid memory architecture* and propose a novel page replacement algorithm, called M-CLOCK, to improve the performance of *hybrid memory architecture* and the lifespan of PCM. In particular, M-CLOCK aims to reduce the number of PCM writes that negatively affect the performance of *hybrid memory architecture*. Experimental results clearly show that M-CLOCK outperforms the state-of-the-art page replacement algorithms in terms of the number of PCM writes and effective memory access time by up to 98% and 9.4 times, respectively.

CCS Concepts: • **Computer systems organization → Embedded systems**;

Additional Key Words and Phrases: Phase change memory, hybrid memory architecture, page replacement algorithm

ACM Reference format:
Minho Lee, Dong Hyun Kang, and Young Ik Eom. 2018. M-CLOCK: Migration-optimized Page Replacement Algorithm for Hybrid Memory Architecture. *ACM Trans. Storage* 14, 3, Article 25 (October 2018), 17 pages.
https://doi.org/10.1145/3216730

1 INTRODUCTION

Recently, several types of new memory technologies such as PCM (PRAM) have been developed. The advent of these new memory devices requires revisiting the page cache layer of the operating system with two major considerations for them to be used as main memory of computer systems. First, PCM has desirable features such as non-volatility, byte-addressability, and in-place update

A preliminary version of this article was presented at the 30th ACM/SIGAPP Symposium on Applied Computing (SAC), Salamanca, Spain, April 2015.
This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (No. NRF-2017R1A2B3004660) and Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT (No. NRF-2015M3C4A7065696).
Authors' addresses: M. Lee and D. H. Kang, Department of Electrical and Computer Engineering, Sungkyunkwan University; emails: {minhozx, dhkangd}@skku.edu; Y. I. Eom (corresponding author), College of Software, Sungkyunkwan University; email: yieom@skku.edu.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2018 Association for Computing Machinery.
1553-3077/2018/10-ART25 $15.00
https://doi.org/10.1145/3216730

(Lee et al. 2010; Kim et al. 2014), which is a main reason why PCM is gaining attention in both academia and industry (Mogul et al. 2009). Second, PCM consumes less power than DRAM, because PCM does not require refresh operations to maintain its contents (Wikipedia 2014; Lee et al. 2016).

Unfortunately, since large-capacity PCM has not been commercially available until now, it cannot totally replace DRAM-based main memory, and so many researchers focus on *hybrid memory architecture* that consists of both DRAM and PCM. *Hybrid memory architecture* can potentially exploit the unique characteristics of each memory device, such as low read/write latency of DRAM and non-volatility of PCM. However, the existing page replacement algorithms, such as LRU, CLOCK, and Linux 2Q, cannot be adopted in this architecture, because they do not consider the two hardware limitations of PCM: (1) PCM suffers from write latency that is about 7 times longer than that of DRAM, and (2) the lifespan of PCM is expected to be significantly shorter than that of DRAM.

Not surprisingly, researchers have suggested various approaches to mitigate the above weaknesses of PCM (Dhiman et al. 2009; Ferreira et al. 2010; Kim et al. 2012; Lee et al. 2013; Qureshi et al. 2009; Seok et al. 2011; Zhou et al. 2009; Dulloor et al. 2016; Yoon et al. 2012; Ramos et al. 2012). Most of the works focused on reducing the number of PCM writes that are crucially concerned with the performance of *hybrid memory architecture* due to asymmetric read/write latency and limited lifespan of PCM. Some studies reduced the write traffic to PCM by employing DRAM as a write-buffer in the *hybrid memory architecture* (Ferreira et al. 2010; Qureshi et al. 2009; Zhou et al. 2009). Although these approaches efficiently reduce the number of PCM writes, they also incur performance degradation, because they should check both DRAM and PCM when the page to be referenced does not exist in DRAM write-buffer. Meanwhile, other studies proposed some page replacement algorithms for *hybrid memory architecture* (Dhiman et al. 2009; Kim et al. 2012; Lee et al. 2013; Seok et al. 2011; Zhong et al. 2014; Chen et al. 2016; Lin et al. 2015; Kang and Eom 2016). Their schemes can be classified into two categories. The first is to reduce the number of PCM writes. To achieve this, they allocate pages to DRAM or PCM after classifying them into write-intensive ones and read-intensive ones. However, they have another computation overhead for classifying memory pages into the two types. The second is for improvement of system performance with strong durability of PCM by using the characteristics such as byte-addressability and non-volatility. In contrast with to the schemes of first category, they try to balance the write count of each cell in PCM, placing dirty pages in PCM.

In this article, we propose a novel page replacement algorithm, called M-CLOCK (Migration-optimized CLOCK), that can improve the performance of *hybrid memory architecture* and extend the lifespan of PCM. This article makes the following specific contributions:

- M-CLOCK has a high page cache hit ratio by extending the traditional CLOCK algorithm that considers temporal locality, by which it reflects the recency of each page and manages pages in the system efficiently.
- M-CLOCK places all faulted pages into DRAM and then secures free space in DRAM by migrating read-intensive pages to PCM or evicting unnecessary pages. In this way, M-CLOCK significantly reduces the number of PCM writes.
- To classify write-intensive pages without computational overhead, M-CLOCK only employs the reference bit and dirty bit that are basic parameters used in the traditional CLOCK algorithm.
- We introduce a *lazy migration* scheme that sometimes allows us to overwrite a page in PCM when write operation occurs on the PCM page, by which it prevents the migration-thrashing problem.

Table 1. The Characteristics of DRAM and PCM

	DRAM	PCM
Non-volatility	X	O
Endurance	N/A	10^7
Read/Write Latency	50/50(ns)	100/350(ns)
Read/Write Energy	0.1/0.1(nJ/bit)	0.2/1.0(nJ/bit)
Static Power	1(W/GB)	0.1(W/GB)

We implemented a prototype of M-CLOCK and compared it with existing page replacement algorithms including CLOCK, CLOCK-Pro, and CLOCK-DWF. The evaluation results clearly show that M-CLOCK significantly reduces the number of PCM writes by up to 99.8% and improves the performance by up to 9.4 times in real systems, compared to CLOCK-DWF.

The remainder of the article is organized as follows. Section 2 briefly presents related work. In Section 3, we describe the detailed design of M-CLOCK. We present the implementation details and evaluation results of M-CLOCK in Section 4 and conclude the article in Section 5.

2 RELATED WORK

PCM is one of the non-volatile memories that, unlike DRAM, does not require memory refresh and provides byte-addressability and in-place update. However, there are two limitations involved in replacing the entire main memory of a computer system with PCM as shown in Table 1 (Eilert et al. 2009). First, PCM has an asymmetric read/write latency; especially, the write latency of PCM is about 7 times longer than that of DRAM. Second, PCM has low endurance, because each PCM cell supports a limited number of write operations, only about 10 million times.

To mitigate the limitations of PCM, previous studies focused on *hybrid memory architecture* that consists of DRAM and PCM and proposed various memory management schemes that can reduce the number of PCM writes and improve the system performace. In this section, we summarize these approaches and discuss their limitations in detail.

The first type of the *hybrid memory architecture* was proposed to reduce the number of PCM writes by exploiting DRAM as upper-layer buffer cache and PCM as main memory (Qureshi et al. 2009; Ferreira et al. 2010; Zhou et al. 2009; Dulloor et al. 2016; Yoon et al. 2012; Ramos et al. 2012). Qureshi et al. (2009) presented a scheme that improves the system performance and extends the lifetime of PCM. They proposed a line level write-back scheme that tracks the writes to DRAM pages at the granularity of cache line and reflects only the dirty cache line on a page in PCM. Ferreira et al. (2010) proposed a buffer cache replacement algorithm that extends the lifetime of PCM by using a clean-preferred victim selection policy and keeping dirty pages in DRAM. Zhou et al. (2009) reduced the total power consumption of the system by adopting the first type of *hybrid memory architecture*. Also, for lifetime extension of PCM, they employed coarse-grained wear leveling mechanism that periodically switches memory segments to balance their write counts. However, their approach differs from ours in that they require hardware implementation and additional management for the DRAM write-buffer. Dulloor et al. (2016) suggested a data-tiering scheme for a heterogeneous memory system that is composed of PCM and small-size DRAM. Their approach provides API to allocate pages on DRAM or PCM according to the page types. However, their approach has some limitations in that their API requires modification of applications with the decision of application developers. Also, they do not consider migration between PCM and DRAM. Yoon et al. (2012) proposed a buffer-caching policy for hybrid DRAM+PCM memories. They focused on allocating data depending on the locality-awareness of the corresponding data for high performance and energy efficiency. For this, their scheme first allocates memory rows

on DRAM and then moves them to PCM after estimating its reuse probability with dynamic threshold adjustment policy when DRAM has no free space. Ramos et al. (2012) introduced a rank-based page migration policy for hybrid main memory that consists of DRAM and PCM. To expand the lifespan of PCM, they employed a hardware-driven page placement policy with the memory controller that monitors page access pattern and decides to migrate pages between DRAM and PCM. But it requires hardware assist to support their page rank decision policy for hybrid main memory.

The second type of the *hybrid memory architecture* used both DRAM and PCM as main memory. For this architecture, several page replacement algorithms have been proposed. To reduce the number of PCM writes and extend its lifetime, they logically partition the main memory, which has fully associative *hybrid memory architecture*, into several regions and manage these regions with their own schemes (Dhiman et al. 2009; Kim et al. 2012; Lee et al. 2013; Seok et al. 2011). Dhiman et al. (2009) proposed a hybrid PRAM and DRAM main memory system with their own scheme. They used an access map that has information on the write count of each page in PRAM. When the write count of a page exceeds the configured threshold, the page is relocated from PRAM to DRAM. However, they did not consider the characteristics of the page (e.g., read- or write-intensive) and simply balanced the write counts of the pages in PRAM. Their scheme also induces an unavoidable PRAM writes by allocating the access map, which is frequently updated, in the PRAM.

Kim et al. (2012) proposed a page replacement algorithm for *hybrid memory architecture* with single chip CPU/GPU, which is logically partitioned into three regions. They reduced the number of PCM writes by first loading a GPU data to a write buffer that is allocated in DRAM and using a write-back policy. When no free page frame exists in the buffer, their scheme migrates the coldest data from the buffer to PCM. However, it includes an overhead caused by adjusting the size of in-DRAM write buffer dynamically according to its current CPU performance. Moreover, it cannot be in general use, because their scheme was designed for a specific environment that employs hybrid memory architecture with single-chip CPU/GPU.

Seok et al. (2011) proposed a page replacement algorithm to reduce the write count of PRAM in *hybrid memory architecture*. Their scheme logically partitions the main memory into four regions: DRAM read, DRAM write, PRAM read, and PRAM write. When a page is referenced, they calculate the weight of the page to determine where to place the page among the four regions. The weight of each page reflects the characteristics of the page: read- or write-intensive. However, it leads to high computation overhead, because the weight of each page is recalculated whenever the page is referenced.

Lee et al. (2013) partitions its set of pages into two regions according to the page characteristics such as the major page access pattern (e.g., read or write). To reduce the number of write operations that occur in PCM, it concentrates on allocating write-intensive pages in DRAM and migrating a page from PCM to DRAM when the write operation occurs on the page in PCM. However, it has two critical problems: (1) It generates unnecessary page migration, because it tries to migrate a page from PCM to DRAM without considering whether the DRAM is full or not. (2) It does not fully use the total capacity of the *hybrid memory architecture* in read- or write-intensive workloads, because it allocates faulted pages only according to the page access pattern.

Zhong et al. (2014) presented a swap mechanism for *hybrid memory architecture* in smartphones. Similar research is suggested by Chen et al. (2016). Although architecture is same with ours, their approaches used NVM only as a swap area to improve the system performance. Lin et al. (2015) and Kang and Eom (2016) proposed page cache algorithms for *hybrid memory architecture*. They focused on improving the performance by reducing the number of storage I/O caused by `fsync` calls, using the non-volatility characteristics of NVM. In contrast with our work, they place dirty pages into NVM for supporting strong durability and eliminating the overhead of fsync calls.

3 THE M-CLOCK ALGORITHM

In this section, we describe the design of our page replacement algorithm, called M-CLOCK, which improves the performance of *hybrid memory architecture* and extends the lifespan of PCM. M-CLOCK aims to reach the following three objectives:

- —Maintain high cache hit ratio by reforming the traditional CLOCK replacement algorithm that considers temporal locality.
- —Minimize the number of PCM writes by placing write-intensive pages on DRAM. It is important to reduce the number of PCM writes, because it negatively affects the performance of *hybrid memory architecture* and the lifespan of PCM.
- —Reduce unnecessary page migration that causes another migration in reverse direction, especially when both PCM and DRAM have no free space. To achieve this, M-CLOCK allows an in-place update in PCM by using *lazy migration* scheme, which eventually contributes to reduce the amount of PCM writes.

The detailed design of M-CLOCK is described in the following two sub-sections: classification of the write-intensive pages in DRAM and *lazy migration*.

3.1 Classification of the Write-intensive Pages in DRAM

For efficient management of pages in DRAM, M-CLOCK exploits a well-known CLOCK algorithm that considers temporal locality (Tanenbaum and Woodhull 1987). The CLOCK algorithm keeps pages in a clock (circular list) and secures a free page by employing a clock-hand (pointer) when no free page exists in the clock. To maintain high hit ratio and mitigate the weaknesses of PCM, we focus on two basic policies for DRAM page management. (1) M-CLOCK always loads the faulted page into DRAM when a page fault occurs and classifies DRAM pages into write-intensive ones and read-intensive ones. (2) When there is no free space in DRAM, M-CLOCK keeps write-intensive pages in DRAM and tries to make free space by migrating a read-intensive pages from DRAM to PCM.

Now, let us explain the details of our page management scheme in DRAM. To efficiently manage DRAM pages, M-CLOCK employs a clock with two clock-hands, *D-hand* and *C-hand*. *D-hand* manages hot-dirty pages, which are frequently referenced by write operations during short periods. However, *C-hand* handles candidate pages, which are unlikely to be referenced by write operations in the near future. M-CLOCK also uses a reference bit and dirty bit of traditional CLOCK replacement algorithm to classify hot-dirty pages and candidate pages. The reference bit of each page describes the recency of the page by setting it whenever the page is referenced. The dirty bit of each page means that the page is updated by write operations. In our scheme, when a page fault occurs, the faulted page is allocated into DRAM and the new page is defined as candidate page as shown in Figure 1(a). At that time, the reference bit of the page is unset until it is referenced again. Meanwhile, if a page fault occurs by a write operation, the dirty bit of the page remains to be set. Otherwise, the dirty bit is unset. Since then, the page is identified as a hot-dirty page according to the page access pattern.

When a candidate page is re-accessed by a write operation, M-CLOCK makes a decision on whether the page is hot-dirty or not. If the reference bit and dirty bit of the page are already set, then the page is considered write-intensive and it is managed by D-hand. Except for this case, M-CLOCK only updates the state of each page by setting the reference bit or dirty bit similarly to the traditional CLOCK replacement algorithm. When no free page exists in DRAM, M-CLOCK secures a free page through two steps. In the first step, M-CLOCK tries to select a page that has the lowest recency among the hot-dirty pages using *D-hand*. If the reference bit of the page pointed by

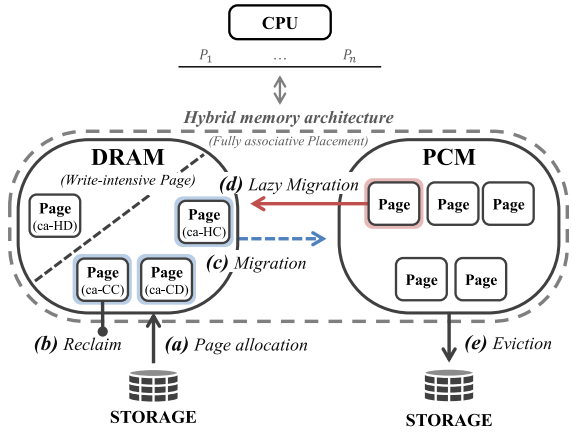

Fig. 1. The overview of M-CLOCK.

Table 2. The Criteria of Victim Page Selection

Type	Reference bit	Dirty bit	Contents
ca-HD	Set	Set	Give a second chance
ca-HC	Set	Unset	Migration to PCM
ca-CD	Unset	Set	Migration to PCM
ca-CC	Unset	Unset	Eviction from DRAM or migration to PCM

D-hand is unset, then the page is updated as candidate page. Otherwise, *D-hand* unsets the reference bit of the page and points to the next hot-dirty page. If *D-hand* dose not find the page whose reference bit is unset while it goes around the list during one circulation, then the second step is performed. In the second step, *C-hand* selects a victim page, which is unlikely to be referenced by write operations in the near future, by inspecting the reference bit and dirty bit of each page while it circulates the list. Then, it evicts the victim page or migrates it from DRAM to PCM. The detail criteria for victim page selection is as follows and summarized in Table 2.

— **ca-HD** (candidate Hot-Dirty)
It means that the page is likely to be referenced again by another write operation, considering that both the reference bit and dirty bit are set. Thus, M-CLOCK considers the page a as write-intensive one that should be kept in DRAM.
— **ca-HC** (candidate Hot-Clean)
This type of page is considered to be potential read-intensive, considering that its reference bit is set and dirty bit is unset. Therefore, by our M-CLOCK scheme, the page is migrated from DRAM to PCM as shown in Figure 1(c).
— **ca-CD** (candidate Cold-Dirty)
Although the dirty bit of the page is set, the page is less likely to be write-intensive, because its reference bit is unset. Thus, the page is migrated from DRAM to PCM as shown in Figure 1(c).
— **ca-CC** (candidate Cold-Clean)
The page is less likely to be read- or write-intensive. It is considered as a depreciated page that is useless to reside in DRAM. If free space exists in PCM, then this page is migrated to PCM; otherwise, it is reclaimed as shown in Figure 1(b).

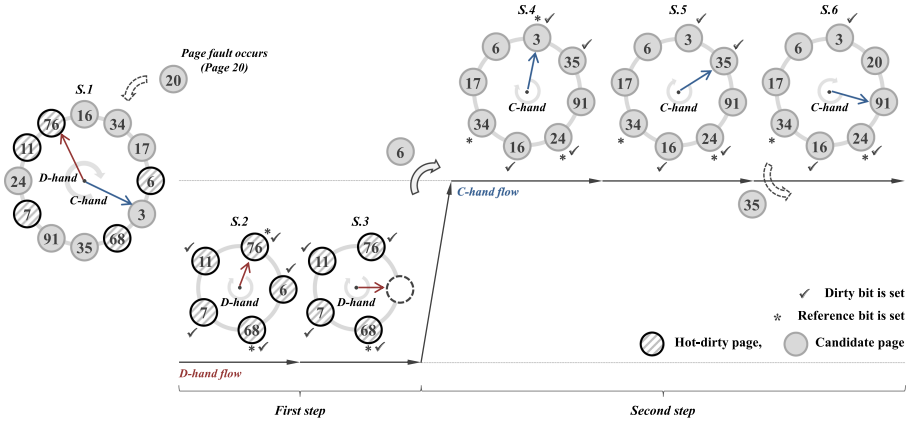

Fig. 2. The example of page migration in DRAM to PCM.

In accordance with the above criteria, if *C-hand* points to the page of ca-HD type in the second step, it clears the reference bit of the page and points to the next page. Otherwise, M-CLOCK migrates the page to PCM. Exceptionally, in the case of a page of ca-CC type, it is evicted from DRAM when both DRAM and PCM is full.

Figure 2 shows an example of the page management of our scheme. Suppose a page fault occurs by *Page 20* when there is no free space in DRAM as shown in Figure 2 S.1. To place the faulted page in DRAM, M-CLOCK first finds a depreciated hot-dirty page via *D-hand* that currently points to *Page 76* as shown in Figure 2 S.2. Since the reference bit of *Page 76* is set, it is cleared, and *D-hand* goes to the next page. In case of *Page 6*, its reference bit is unset. Thus, the page is updated to candidate page as shown in Figure 2 S.3. After updating hot-dirty page to candidate page, *C-hand* tries to evict or migrate a victim page that is least likely to be write-intensive. *C-hand* points to *Page 3* whose reference bit and dirty bit are set. According to the M-CLOCK scheme, the reference bit of *Page 3* is cleared and then *C-hand* points to the next page (*Page 35*). In the case of *Page 35*, although the dirty bit is set, it is unlikely to be write-intensive, because its reference bit is unset. Consequentially, *Page 35* is migrated from DRAM to PCM.

3.2 Page Eviction and Lazy Migration in PCM

To efficiently manage the pages in PCM, M-CLOCK employs another clock with a clock-hand (*P-hand*), apart from that of DRAM. In PCM, M-CLOCK focuses on two operations. One is the page eviction that is performed similar to the traditional CLOCK replacement algorithm (Figure 1(e)). When there is no free space in PCM, M-CLOCK scans a list of pages on PCM to select a victim page whose reference bit is unset. If the reference bit of the page pointed *P-hand* is set, then it is cleared and then *P-hand* points the next page in the list of pages on PCM. Otherwise, the page is evicted from PCM to storage.

Another is the *lazy migration* to mitigate the weaknesses of PCM by reducing the amount of PCM writes and unnecessary page migrations between DRAM and PCM (Figure 1(d)). *Lazy migration* is triggered when a write operation occurs on a page in PCM. To carry out *lazy migration*, M-CLOCK uses an additional bit, called *lazy bit*, on each page in PCM. The detailed operation of *lazy migration* is described with the pseudo-code in Algorithm 1. When a write operation occurs on a page in PCM, M-CLOCK checks whether a free page exists in DRAM (#1). If a free page exists in DRAM, then M-CLOCK migrates the page accessed by the write operation from PCM to DRAM, and then the write operation is carried out in DRAM. Otherwise, M-CLOCK checks the *lazy bit* of

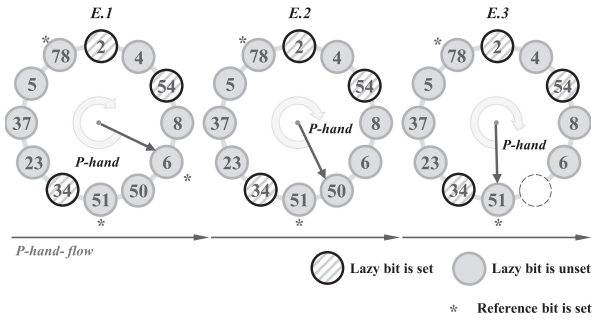

Fig. 3. The example of page eviction in PCM.

ALGORITHM 1: Basic algorithm of lazy migration

```
/* When the page is referenced in PCM*/
procedure LAZY_MIGRATION(page p, page access type op)
    if op is 'R' then
        p's reference_bit is set
    else
        if free page exists in DRAM then                    (#1)
            migrate p from PCM to DRAM
        else                                                (#2)
            if p's lazy_bit is set then
                migrate p from PCM to DRAM
            else
                overwrite a requested data on p
                p's reference_bit and p's dirty_bit are set
                p's lazy_bit is set
            end if
        end if
    end if
end procedure
```

the page to decide whether or not to migrate the page (#2). If *lazy bit* is unset, then M-CLOCK just overwrites the data accessed by the write operation set the *lazy bit* of the page in PCM. Otherwise, M-CLOCK considers the page as a hot-dirty page and migrates the page from PCM to DRAM.

Figure 3 depicts an example of page eviction in PCM. In our scheme, when DRAM is full, some of its pages are migrated to PCM. Such page migrations may lead to a case where there is no free space in PCM. Thereby, page eviction is required to make free space in PCM. Figure 3 E.1 shows that *P-hand* points to *Page 6*. Since reference bit of *Page 6* is set, it is cleared by *P-hand*, and then *P-hand* points to the next page (*Page 50*). Now, the reference bit of *Page 50* is unset, and *Page 50* is evicted from PCM to storage. An example of *lazy migration* is shown in Figure 4. Suppose DRAM is already full and write operation occurs on *Page 48* in PCM. M-CLOCK checks the *lazy bit* of *Page 48*. Since the *lazy bit* of *Page 48* is unset as shown in Figure 4 L.1, M-CLOCK just overwrites the data accessed by the write operation on *Page 48* and then sets the *lazy bit* of *Page 48*. If write operation occurs once again on *Page 48*, then M-CLOCK considers the page as write-intensive and migrates the page from PCM to DRAM by the *lazy migration* scheme.

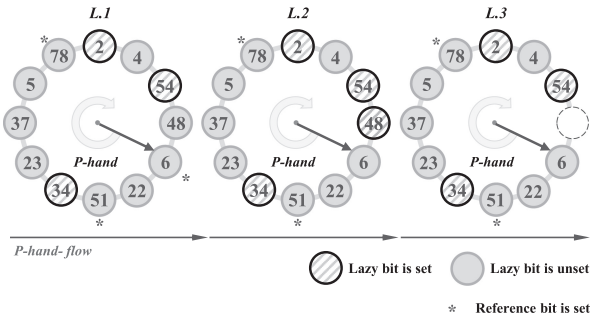

Fig. 4. The example of lazy migration.

4 EVALUATION

In this section, we evaluate the performance of M-CLOCK to answer the following questions:

- —What is the effectiveness of M-CLOCK in classifying write-intensive pages from other pages in DRAM?
- —What are the performance benefits of *lazy migration* in terms of the number of unnecessary migrations and the amount of PCM writes?
- —Can M-CLOCK bring the performance benefits in real systems that are equipped with DRAM and PCM?
- —How much power efficiency is improved with M-CLOCK in *hybrid memory architecture* that consists of DRAM and PCM?

We first describe the experimental setup and then present the experimental results, comparing M-CLOCK with the existing page replacement algorithms, including CLOCK, CLOCK-Pro (Jiang et al. 2005), and CLOCK-DWF.

4.1 Experimental Setup

To verify the effectiveness of M-CLOCK, we used an evaluation framework that is based on the trace-driven simulator, but some experiments are performed on real PCM-based board for more concise evaluation results. For the evaluation framework, we first developed the trace-driven simulator to emulate the *hybrid memory architecture* that consists of DRAM and PCM. We also implemented M-CLOCK and the existing page replacement algorithms, including CLOCK, CLOCK-Pro, and CLOCK-DWF.

Since CLOCK and CLOCK-Pro have been suggested for DRAM-based memory architecture, they only focus on the high hit ratio by considering temporal locality about the allocated pages in memory. Thus, for comparison with M-CLOCK in fully associative *hybrid memory architecture*, the eviction flow of them have to be modified so that they migrate pages from DRAM to PCM when page fault occurs and there is no free space in DRAM. In our experiments, to confirm the effectiveness of temporal locality with respect to the classification of write-intensive pages, CLOCK and CLOCK-Pro are adopted as page replacement algorithms for DRAM in *hybrid memory architecture*. With regard to the allocated pages in PCM, they are managed just by CLOCK algorithm based on the temporal locality, which is similar to the CLOCK-DWF and M-CLOCK, while taking another method for the page migration policies from PCM to DRAM, as described in Section 3.

In case of CLOCK-DWF that needs a parameter setting, we configured the overlooked rotation count of CLOCK-DWF to 8 as recommended by the authors of the article. Moreover, we modified `cachegrind` of `valgrind` tools (Nethercote and Seward 2003) and then collected memory traces

Table 3. The Characteristics of Each Workload

Workload	Memory footprint (KB)	Ratio of operations (Write : Read)
freecell	21,300	1 : 3.4
shotwell	42,728	1 : 1.5
gnuplot	4,540	4.1 : 1
gqview	58,964	1 : 1.3

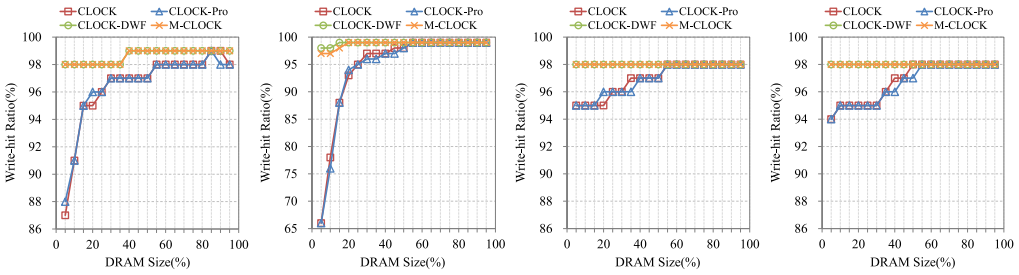

Fig. 5. The write-hit ratio in DRAM.

using it while running four applications; `freecell`, `shotwell`, `gnuplot`, and `gqview`. The characteristics of the gathered memory traces are depicted in Table 3. We used the memory traces as input and configured the size of DRAM from 5% to 95% of the total size of the *hybrid memory architecture* of which capacity is equal to the memory footprint of the workloads.

4.2 Write Hit Ratio and PCM Writes

In *hybrid memory architecture*, classifying write-intensive pages and allocating the pages in DRAM is one of the important issues. It is because the negligent classification of write-intensive pages induces performance degradation and reduces the lifespan of PCM due to the weaknesses of PCM such as high write latency and low endurance. In this subsection, we first present the write-hit ratio in DRAM and the number of PCM writes for each page replacement algorithm. Moreover, we verify the effectiveness of M-CLOCK, comparing it with other page replacement algorithms in respect of memory utilization.

Figure 5 shows the write-hit ratio in DRAM, comparing M-CLOCK with CLOCK, CLOCK-Pro, and CLOCK-DWF. The higher write-hit ratio means that more write-intensive pages are held in DRAM during the workload execution. As shown in Figure 5, M-CLOCK outperforms the other page replacement algorithms by up to 34%. It is because M-CLOCK efficiently keeps write-intensive pages in DRAM by employing only the reference bit and dirty bit in the architecture. The write-hit ratio of CLOCK-DWF is similar to that of M-CLOCK, because it considers not only temporal locality but also write frequency to hold write-intensive pages in DRAM. However, CLOCK-DWF needs more parameter setting for each page than M-CLOCK in classifying write-intensive pages from other pages, which leads to computational overhead.

Figure 6 shows the number of PCM writes of M-CLOCK normalized to that of CLOCK-DWF. Since CLOCK and CLOCK-Pro have a large number of PCM writes in that they do not migrate the write-intensive pages from PCM to DRAM, they are excluded from the graphs in Figure 6. When the percentage of DRAM size is small in the *hybrid memory architecture*, the number of PCM writes of M-CLOCK is similar to that of CLOCK-DWF. However, as the percentage of DRAM size

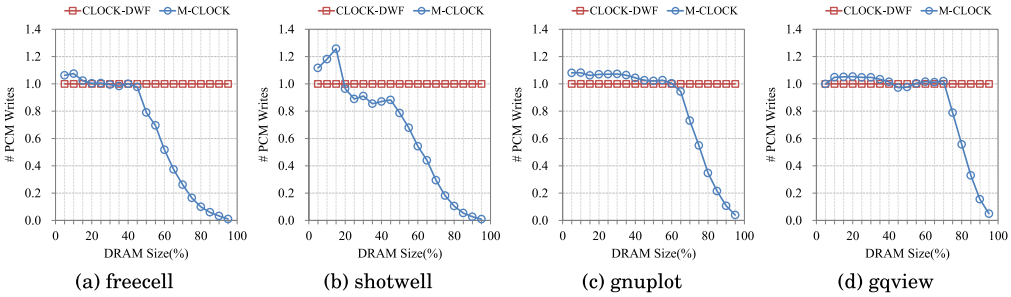

Fig. 6. The number of PCM writes of M-CLOCK normalized to that of CLOCK-DWF.

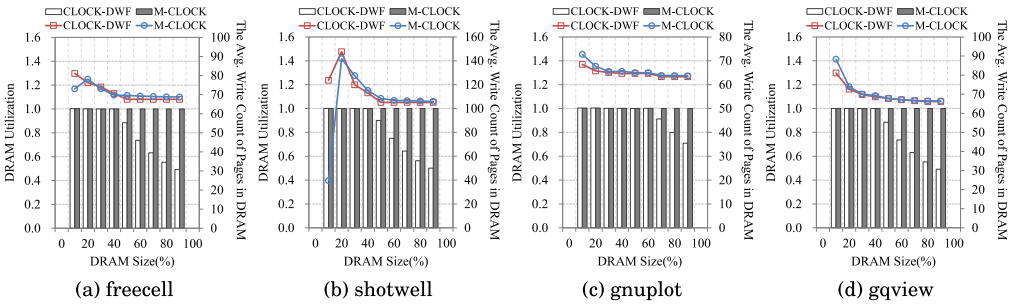

Fig. 7. DRAM utilization and the average write count of pages in DRAM.

increases, M-CLOCK further reduces the number of PCM writes, by up to about 98%, compared to CLOCK-DWF.

To understand why M-CLOCK performs better, we measured not only DRAM utilization but also average write count of dirty pages in DRAM for each page replacement algorithm, as shown in Figure 7. Despite CLOCK-DWF maintains high write-hit ratio in DRAM, it does not fully use the capacity of DRAM for all workloads. It is because CLOCK-DWF just allocates the faulted page in DRAM or PCM according to the page access type when page fault occurs. For example, if the workload is read-intensive, CLOCK-DWF uses only the space of PCM although DRAM has free space. Also, when the workload is write-intensive, CLOCK-DWF uses only the space of DRAM although PCM has free space. However, M-CLOCK fully utilizes the memory space of the *hybrid memory architecture* regardless of the type of workload (e.g., read-intensive or write-intensive). The major reason is that M-CLOCK first loads all faulted pages into DRAM and then migrates some pages to PCM when no free page exists in DRAM. By doing so, M-CLOCK can maintain high hit ratio in DRAM and efficiently hold write-intensive pages in DRAM with only two bits: reference bit and dirty bit.

4.3 The Effect of Lazy Migrations

Since write operations are commonly requested at various sizes of byte granularity, page migration from PCM to DRAM should be carefully decided. The hasty page migration for reducing the number of PCM writes may generate the migration-thrashing problem. For example, suppose a write operation occurs on a page in PCM. If both DRAM and PCM has no free space in *hybrid memory architecture*, then a page migration from PCM to DRAM causes another migration in reverse direction. In this case, the number of PCM writes is equal to the case of overwriting the page in PCM. Also, the amount of data overwritten to PCM is smaller (<4KB) than in the case of

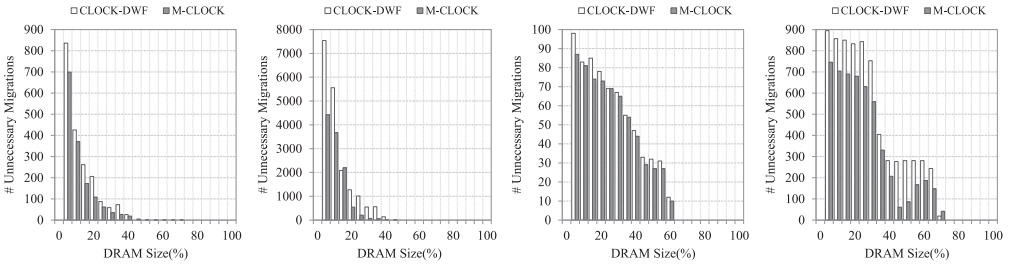

Fig. 8. The number of unnecessary migrations between DRAM and PCM.

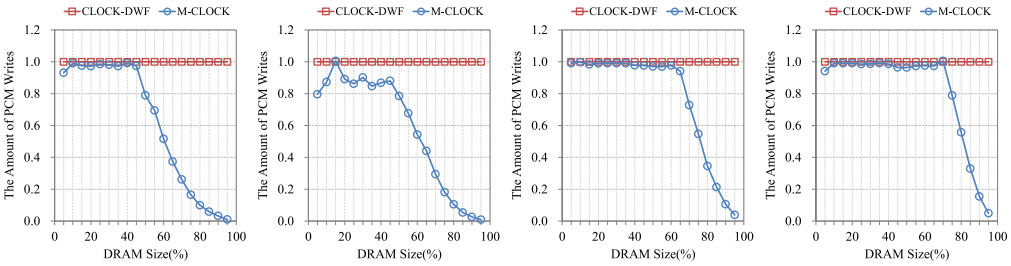

Fig. 9. The amount of PCM writes of M-CLOCK normalized to that of CLOCK-DWF.

migrating a page from DRAM to PCM (4KB). For this reason, we measured the number of unnecessary page migrations and the amount of PCM writes for each page replacement algorithm.

Figure 8 shows the number of unnecessary page migrations of M-CLOCK and CLOCK-DWF. As shown in Figure 8, M-CLOCK mitigates the migration-thrashing problem in that it reduces the number of unnecessary page migrations by up to 88%, compared with CLOCK-DWF. It is because M-CLOCK copes with the write operations in PCM according to *lazy migration*. Figure 9 shows that the amount of PCM writes of M-CLOCK normalized to that of CLOCK-DWF. Although M-CLOCK has slightly larger number of PCM writes than CLOCK-DWF when the percentage of DRAM size is small as shown in Figure 6, M-CLOCK reduces the amount of PCM writes by up to 8% compared with CLOCK-DWF as shown in Figure 9. Moreover, the gap in the amount of PCM writes becomes considerably large, by up to 99% when the percentage of DRAM size increases. In other words, the amount of PCM writes is not proportional to the number of PCM writes and differently affects the performance of *hybrid memory architecture*. In case of CLOCK-DWF, to reduce the number of PCM writes, it does not overwrite data when write operation occurs on a page in PCM. Instead, it migrates the page from PCM to DRAM and then carries out the write operation. However, such page migration may generate the migration-thrashing problem. However, M-CLOCK allows in-place update when write operation occurs on a page in PCM, and then it migrates the page from PCM to DRAM when write operation occurs again (*lazy migration*). For this reason, M-CLOCK can reduce the number of unnecessary page migrations and have small average amount of PCM writes compared with CLOCK-DWF.

In addition, to examine the effect of lazy migration in more detail, we performed experiments while varying the permissible number of write operations on the allocated pages on PCM from 1 to 16 and then measured the number of PCM writes and unnecessary migrations between PCM and DRAM. For the experiments, we configured the size of DRAM to 20% of the total size of the

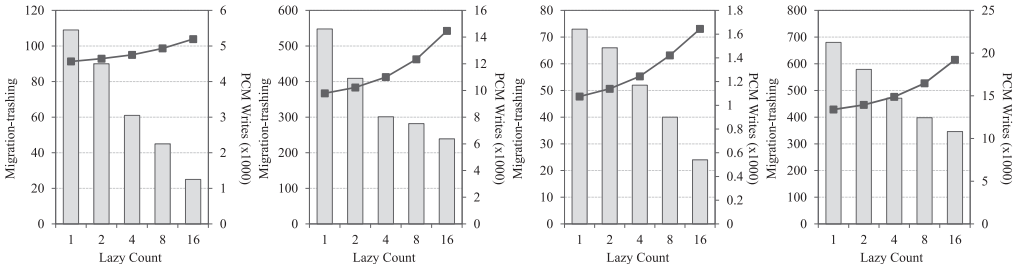

Fig. 10. The number of PCM writes and unnecessary page migrations of M-CLOCK.

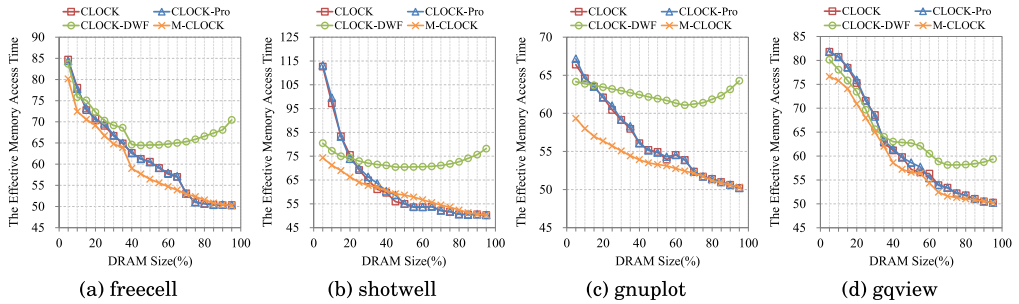

Fig. 11. The effective memory access time.

hybrid memory architecture and used a countable value, called lazy count, instead of lazy bit. Figure 10 shows that there is a tradeoff between the lazy count and the lifetime of PCM. As shown in Figure 10, the number of unnecessary migrations in *hybrid memory architecture* drops by up to 77.1% when the lazy count increases, whereas the number of PCM writes considerably increases by up to 53.1% in this case. From the endurance point of view, to reduce the number of unnecessary migrations between PCM and DRAM, allowing the more number of write operations on the allocated pages on PCM is inefficient in that the reduced number of unnecessary migrations is significantly smaller than that of PCM writes. Thus, the experimental results demonstrate that the lazy bit is more suitable than the lazy count for lazy migration of M-CLOCK.

4.4 Effective Memory Access Time

To demonstrate the relationship between PCM writes and the performance of *hybrid memory architecture*, we compared effective memory access time of the *hybrid memory architecture* for each page replacement algorithm, as shown in Figure 11. The effective memory access time can be described as Equation (1) with the parameters in Table 4,

$$\begin{aligned} T_E &= \alpha \times T_M + (1-\alpha) \times T_F \\ T_M &= \beta \times T_P + (1-\beta) \times T_D \\ T_D &= 50, \\ T_P &= (N_{P_r} \times L_{P_r} + N_{P_w} \times L_{P_w})/(N_{P_r} + N_{P_w}) \end{aligned} \tag{1}$$

For the detail comparison in respect of memory access time, T_F is excluded in Figure 11. T_M can be calculated with T_D and T_P. T_D is always 50ns, because DRAM has symmetric read/write latency (see Table 1), whereas T_P should be calculated with the number of PCM writes (N_{P_w}) and reads

Table 4. Description of Parameters for Equation (1)

Value	Description
T_E	Effective memory access time
T_M	Average memory access time for the *hybrid memory architecture with PCM and DRAM*
T_F	Page fault processing time
T_P, T_D	Average PCM or DRAM access time
α, β	Memory hit ratio, probability that the page exists in PCM
N_{P_w}, N_{P_r}	Number of PCM writes/reads
L_{P_w}, L_{P_r}	Write/Read latency of PCM

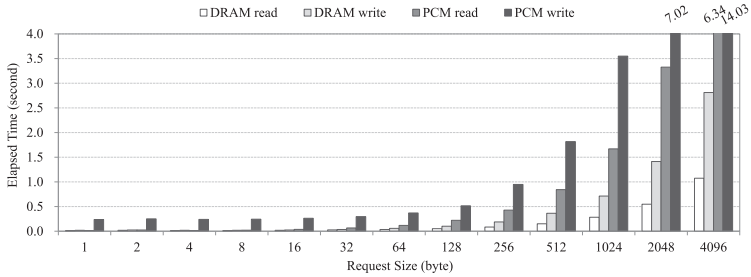

Fig. 12. The memory access time of DRAM and PCM.

(N_{P_r}) due to the asymmetric read/writs latency of PCM, of which the read latency (L_{P_r}) differs from its write latency (L_{P_w}). As shown in Figure 11, M-CLOCK has lower effective memory access time by up to 34% compared with other page replacement algorithms. Since M-CLOCK well holds write-intensive pages in DRAM, T_E of M-CLOCK is relatively lower than that of CLOCK and CLOCK-Pro whose T_P is higher in that they have larger number of PCM writes. Although T_P of CLOCK-DWF is similar to that of M-CLOCK, T_E of CLOCK-DWF is higher than M-CLOCK, because it sometimes shows low page cache hit ratio according to the characteristics of the workloads.

As aforementioned in Section 2, the read and write latency of PCM is longer than that of DRAM by about 2 times and 7 times, respectively. However, the performance gap is different in a non-volatility emulation board equipped with Xilinx Zynq XC7Z020, 1GB of PS-DRAM, and 2GB of PL-DRAM (Chung et al. 2011; Lee et al. 2014). Since PL-DRAM is connected with non-volatile memory (NVM) power and has lower latency than PS-DRAM in this board, its performance can be considered as that of PCM. Figure 12 shows the performance of DRAM and PCM in terms of latency after completing 10,000 access requests to each memory. In the real PCM-based board, the read and write latency of PCM is longer than that of DRAM by 3.5 times and 7.9 times on average (up to 6 times and 12 times), respectively. Based on these results, to measure the effective memory access time in real PCM-based board, we implemented a trace replay tool that allocates and migrates pages between DRAM and PCM according to the page type, in real PCM-based board.

Figure 13 shows that the effective memory access time of M-CLOCK outperforms that of CLOCK-DWF by up to 9.4 times. It is because CLOCK-DWF wastes the capacity of DRAM and only utilizes the capacity of PCM though there is no free space in PCM when page fault occurs by read requests. The effective memory access time of CLOCK and CLOCK-Pro is similar to that of M-CLOCK, because they carry out write operations by small size of byte granularity in PCM. However, they generate a considerable number of PCM writes (up to 7.7 times) due to the absence

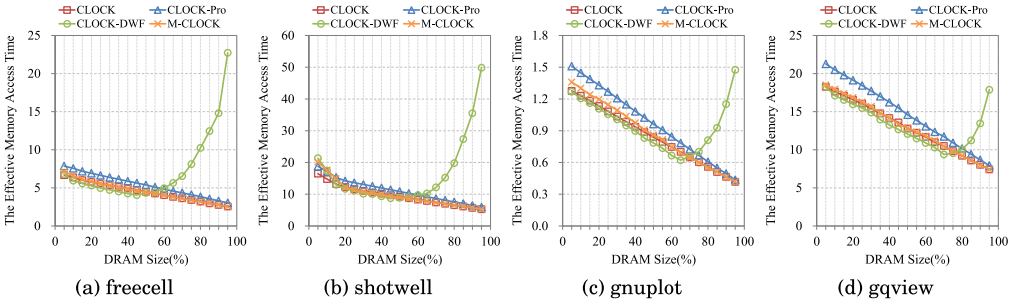

Fig. 13. The effective memory access time in real board.

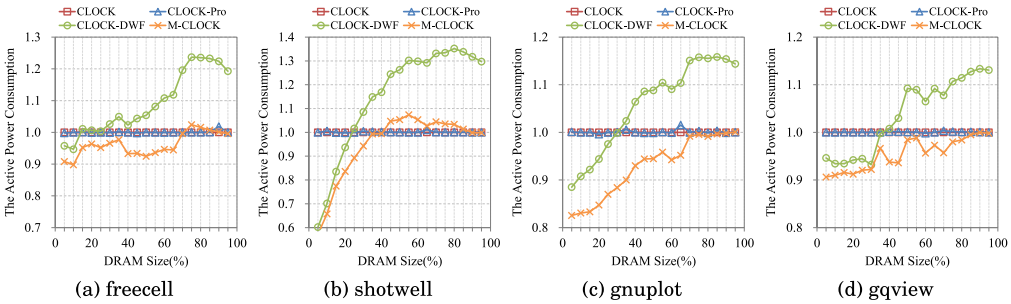

Fig. 14. The active power consumption of M-CLOCK normalized to that of CLOCK.

of classification of write-intensive pages for *hybrid memory architecture*, which negatively affects the lifespan of PCM.

4.5 Power Consumption

In this subsection, we evaluate the benefit of M-CLOCK in respect of power consumption. PCM has low static power consumption due to its non-volatility, whereas it consumes more power than DRAM when read or write operation occurs on a page in PCM. Thus, reducing the number of PCM writes affects the power consumption of *hybrid memory architecture*,

$$\begin{aligned} P_t &= P_s + P_a \\ &= (P_{s_{dram}} + P_{s_{pcm}}) + (P_{a_{dram}} + P_{a_{pcm}}) \end{aligned} \tag{2}$$

$$P_a = (N_r \times E_r + N_w \times E_w)/(N_r \times L_r + N_w \times L_w)$$
$$P_s = memory\_size \times unit\_static\_power.$$

Equation (2) shows the total power consumption of the *hybrid memory architecture*, which is the sum of the static power (P_s) and active power (P_a). The static power consumption is interrelated to the memory size and *unit static power* of each memory device. The active power consumption originates from the read/write operations on each memory device. N is the number of reads/writes, and E is the energy consumption that is generated when read/write operation occurs in each memory device. L is the read/write latency of each memory. Since the static power consumption relies on the size of each memory device, Figure 14 only shows the active power consumption for each page replacement algorithm normalized to that of CLOCK. In case of M-CLOCK, power consumption is saved by up to 40% compared with other page replacement algorithms. It is because

M-CLOCK reduces the number of PCM writes by inducing most write operations to occur on DRAM as aforementioned.

5 CONCLUSIONS

PCM-based memory architecture has been extensively studied for employing it as a main memory due to its attractive features, such as non-volatility, byte-addressability, and in-place update. However, there are two hardware limitations for whole main memory to be replaced with PCM in spite of its attractive characteristics. First, PCM suffers from write latency, because the write operation is about 7 times slower than the read operation and the latency of write operation primarily depends on the amount of writes. Second, the lifetime of PCM is expected to be significantly shorter than that of DRAM, because each PCM cell supports a limited number of write operations, only a million times. Furthermore, since current memory technology does not resolve the capacity problem of PCM, many researchers focus on *hybrid memory architecture* that consists of DRAM and PCM.

In this article, we propose a novel page replacement algorithm, called M-CLOCK, for *hybrid memory architecture*, considering the weaknesses of PCM. M-CLOCK concentrates on classifying write-intensive pages among the DRAM pages that are allocated when page fault occurs. Moreover, M-CLOCK keeps write-intensive pages in DRAM and, if necessary, secures and makes free pages in DRAM by migrating read-intensive pages to PCM or evicting unnecessary pages that are unlikely to be referenced in the near future. M-CLOCK also uses *lazy migration*, which delays migration (from PCM to DRAM) of the page requested by write operation, allowing us to overwrite the data on the pages in PCM. By doing so, M-CLOCK can reduce unnecessary migrations and the amount of PCM writes.

Experimental results show that M-CLOCK has high write-hit ratio, by up to 34%, compared with the existing page replacement algorithms for all workloads used in our experiments. M-CLOCK also reduces the number of PCM writes by up to 98%. Furthermore, M-CLOCK improves the performance of *hybrid memory architecture* in terms of effective memory access time by up to 9.4 times compared with other page replacement algorithms. By reducing the number of PCM writes and the amount of PCM writes, we can reduce the power consumption by up to 20%.

REFERENCES

Xianzhang Chen, Edwin H.-M. Sha, Weiwen Jiang, Qingfeng Zhuge, Junxi Chen, Jiejie Qin, and Yuansong Zeng. 2016. The design of an efficient swap mechanism for hybrid DRAM-NVM systems. In *Proceedings of the IEEE International Conference on Embedded Software (EMSOFT'16)*. ACM, 1–10.

Hoeju Chung, Byung Hoon Jeong, and Byungjun Min. 2011. A58nm 1.8V 1Gb PRAM with 6.4MB/s program BW. In *Proceedings of the IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC'11)*. IEEE, 500–502.

Gaurav Dhiman, Raid Ayoub, and Tajana Rosing. 2009. PDRAM: A hybrid PRAM and DRAM main memory system. In *Proceedings of the Design Automation Conference (DAC'09)*. IEEE, 664–669.

Subramanya R. Dulloor, Amitabha Roy, Zheguang Zhao, Narayanan Sundaram, Nadathur Satish, Rajesh Sankaran, Jeff Jackson, and Karsten Schwan. 2016. Data tiering in heterogeneous memory systems. In *Proceedings of the European Conference on Computer Systems (EuroSys'16)*. ACM, 1–16.

Sean Eilert, Mark Leinwander, and Giuseppe Crisenza. 2009. Phase change memory: A new memory enables new memory usage models. In *Proceedings of the International Memory Workshop (IMW'09)*. IEEE, 1–2.

Alexandre P. Ferreira, Miao Zhou, Santiago Bock, Bruce Childers, Rami Melhem, and Daniel Mossé. 2010. Increasing PCM main memory lifetime. In *Proceedings of the Design, Automation and Test in Europe (DATE'10)*. European Design and Automation Association, 914–919.

Song Jiang, Feng Chen, and Xiaodong Zhang. 2005. CLOCK-Pro: An effective improvement of the clock replacement. In *Proceedings of the USENIX Annual Technical Conference (ATC'05)*. USENIX, 323–336.

Dong Hyun Kang and Young Ik Eom. 2016. FSLRU: A page cache algorithm for mobile devices with hybrid memory architecture. *IEEE Trans. Cons. Electron.* 62, 2 (2016), 136–143.

Dongki Kim, Sungkwang Lee, Jaewoong Chung, Dae Hyun Kim, Dong Hyuk Woo, Sungjoo Yoo, and Sunggu Lee. 2012. Hybrid DRAM/PRAM-based main memory for single-chip CPU/GPU. In *Proceedings of the Design Automation Conference (DAC'12)*. ACM, 888–896.

Hyojun Kim, Sangeetha Seshadri, Clement L. Dickey, and Lawrence Chiu. 2014. Evaluating phase change memory for enterprise storage systems: A study of caching and tiering approaches. In *Proceedings of the USENIX Conference on File and Storage Technologies*. USENIX, 33–45.

Benjamin C. Lee, Ping Zhou, Jun Yang, Youtao Zhang, Bo Zhao, Engin Ipek, Onur Mutlu, and Doug Burger. 2010. Phase change technology and the future of main memory. *IEEE Micro* 30, 1 (2010), 131–141.

Eunji Lee, Julie Kim, Hyokyung Bahn, and Sam H. Noh. 2016. Reducing write amplification of flash storage through cooperative data management with NVM. In *Proceedings of the IEEE International Conference on Massive Data Storage Systems and Technology (MSST'16)*. IEEE, 1–6.

Soyoon Lee, Hyokyung Bahn, and S. Noh. 2013. CLOCK-DWF: A write-history-aware page replacement algorithm for hybrid PCM and DRAM memory architectures. *IEEE Trans. Comput.* 63, 9 (2013), 1–14.

Taemin Lee, Dongki Kim, and Hyunsun Park. 2014. FPGA-based prototyping systems for emerging memory technologies. In *Proceedings of the IEEE International Symposium on Rapid System Prototyping (RSP'14)*. IEEE, 115–120.

Ye-jyun Lin, Chia-Lin Yang, Hsiang-Pang Li, and Cheng-Yuan Michael Wang. 2015. A buffer cache architecture for smartphones with hybrid DRAM/PCM memory. In *Proceedings of the IEEE Non-Volatile Memory System and Applications Symposium (NVMSA'15)*. IEEE, 1–6.

Jeffrey C. Mogul, Eduardo Argollo, Mehul Shah, and Paolo Faraboschi. 2009. Operating system support for NVM+DRAM hybrid main memory. In *Proceedings of the USENIX Hot Topics in Operating Systems (HotOS'09)*. USENIX, 1–5.

Nicholas Nethercote and Julian Seward. 2003. Valgrind: A program supervision framework. *Electr. Not. Theoret. Comput. Sci.* 89, 2 (2003), 44–66.

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

Luiz E. Ramos, Eugene Gorbatov, and Ricardo Bianchini. 2012. Page placement in hybrid memory systems. In *Proceedings of International Conference on Supercomputing (ICS'12)*. ACM, 95–95.

Hyunchul Seok, Youngwoo Park, Ki-Woong Park, and Kyu Ho Park. 2011. Efficient page caching algorithm with prediction and migration for a hybrid main memory. *ACM SIGAPP Appl. Comput. Rev.* 11, 4 (2011), 38–48.

Andrew S. Tanenbaum and Albert S. Woodhull. 1987. *Operating Systems: Design and Implementation*. Prentice-Hall Englewood Cliffs, NJ.

Wikipedia. 2014. Memory Refresh. Retrieved from http://en.wikipedia.org/wiki/Memory_refresh/.

HanBin Yoon, Justin Meza, Rachata Ausavarungnirun, Rachael A. Harding, and Onur Mutlu. 2012. Row buffer locality aware caching policies for hybrid memories. In *Proceedings of the International Conference on Computer Design (ICCD'12)*. IEEE, 337–344.

Kan Zhong, Tianzheng Wang, Xiao Zhu, Linho Long, Duo Liu, Weichen Liu, Zili Shao, and Edwin H.-M. Sha. 2014. Building high-performance smartphones via non-volatile memory: The swap approach. In *Proceedings of the IEEE International Conference on Embedded Software (EMSOFT'14)*. ACM, 1–10.

Ping Zhou, Bo Zhao, Jun Yang, and Youtao Zhang. 2009. A durable and energy efficient main memory using phase change memory technology. *ACM SIGARCH Comput. Arch. News* 37, 3 (2009), 14–23.

Received February 2017; revised March 2018; accepted April 2018