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

Minho Lee, Dong Hyun Kang, Junghoon Kim, Young Ik Eom
College of Information and Communication Engineering, Sungkyunkwan University
Suwon, 440-746, Korea
{minhozx, kkangsu, myhuni20, yieom}@skku.edu

ABSTRACT

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. 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 paper, to mitigate the above hardware limitations of PCM, we revisit the page cache layer for the *hybrid memory architecture*. We also 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 34%, respectively.

Categories and Subject Descriptors

B.3.2 [**Memory Structures**]: Design Styles Cache memories; D.4.2 [**Software Engineering**]: Operating Systems—*Storage Management*

General Terms

Page replacement algorithm

Keywords

Phase Change Memory, PCM, Hybrid Memory Architecture, Page Replacement Algorithm

SAC'15 April 13-17, 2015, Salamanca, Spain.

http://dx.doi.org/10.1145/2695664.2695675.

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, in order 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 [6], which is a main reason why PCM is gaining attention in both academia and industry. Second, PCM consumes less power than DRAM, because PCM does not require refresh operations [12]. 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, 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 [1, 3, 5, 7, 9, 10, 13]. 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* [3, 9, 13]. 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* [1, 5, 7, 10]. To reduce the number of PCM writes, their schemes 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.

In this paper, 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 paper 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 *lazy migration* scheme that sometimes allows to overwrite a page when write operation occurs on the page in PCM, by which it prevents the migration-thrashing problem.

The remainder of the paper 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 paper in Section 5.

2. RELATED WORK

PCM is one of the non-volatile memories that, unlike DRAM, does not require the 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 [2]. 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 ten million times.

	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)

Table 1: The characteristics of DRAM and PCM

To mitigate the limitations of PCM, previous studies focused on *hybrid memory architecture* that consists of DRAM and PCM, and proposed various approaches for reducing the number of PCM writes. Qureshi et al. [9] presented *hybrid memory architecture* that consists of PCM and DRAM, which is used as an upper-layer buffer cache. They reduced the write traffic to PCM by holding the write-intensive pages in DRAM write-buffer. However, their approach differs from ours in that it requires hardware implementation and additional management for DRAM write-buffer.

Dhiman et al. [1] proposed a hybrid PRAM and DRAM main memory system with their own scheme. They used the 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. [5] 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 use 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.

Lee et al. [7] partitions its set of pages into two regions according to the page characteristics such as the major page access type (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 *hybrid memory architecture* in read- or write-intensive workloads because it allocates faulted pages only according to the page access type.

Seok et al. [10] 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 such as read- or write-intensive. However, it leads to high computation overhead because the weight of each page is recalculated whenever the page is referenced.

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 two objectives.

- 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 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.

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 [11]. The CLOCK algorithm keeps pages in a clock (circular list) and makes a free page by employing a clock-hand (pointer) when no free page exists in the clock.

Now, let us explain the details of our page classification scheme in DRAM. To efficiently classify DRAM pages into write-intensive ones and read-intensive ones, 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. On the other hand, C-hand handles candidate pages, which are unlikely to be referenced by write operations in the future. M-CLOCK also uses a reference bit and dirty bit of the traditional CLOCK replacement algorithm to classify pages into two types: 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 page is updated by write operation. The two bits are separately set when M-CLOCK first loads a faulted page according to page access type such as read or write, and then the faulted page become hot-dirty page as shown in Figure 1a.

If 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, the page is considered as write-intensive one 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, D-hand tries to select a page that has the lowest recency among hot-dirty page. If the reference bit of the page pointed by D-hand is unset, 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, the second step is performed. In the second step, C-hand selects a victim page by inspecting the reference bit and dirty bit of each page while it circulates the clock. The criteria for victim page selection is as follows and summarized in Table 2.

	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

Table 2: The criteria of victim page selection

- **ca-HD** (candidate Hot-Dirty): It means that the page is likely to be referenced again by another write operation, considering both its reference bit and dirty bit are set. Thus, M-CLOCK considers the page 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 its reference bit is set and its dirty bit is unset. Therefore, by our M-CLOCK scheme, the page is migrated from DRAM to PCM as shown in Figure 1c.
- **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 1c.
- **ca-CC** (candidate Cold-Clean): The page is less likely to be read- or write-intensive. It is considered as depreciated page that is useless to reside in DRAM. For this reason, this page is reclaimed as shown in Figure 1b.

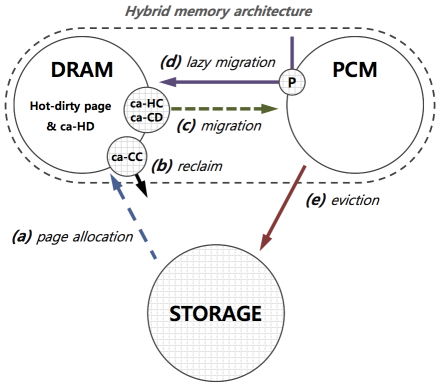


Figure 1: The overview of M-CLOCK

3.2 Lazy Migration

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 *lazy migration* for reducing the amount of PCM writes and unnecessary page migrations (Figure 1d). Another is the page eviction that is performed similar to the traditional CLOCK replacement algorithm (Figure 1e). *Lazy migration* is triggered when write operation occurs on a page in PCM. To carry out *lazy migration*, M-CLOCK uses 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.

Algorithm 1 Basic algorithm of lazy migration

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

When the 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, M-CLOCK migrates the page requested by write operation from PCM to DRAM, and then the write operation is carried out. Otherwise, M-CLOCK

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

4. EVALUATION

To emulate the *hybrid memory architecture* that consists of DRAM and PCM, we performed trace-driven simulations. We also implemented M-CLOCK and the existing page replacement algorithms, including CLOCK, CLOCK-Pro [4], and CLOCK-DWF. For comparison, CLOCK and CLOCK-Pro are adopted as page replacement algorithms for DRAM in *hybrid memory architecture*. 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 paper. In addition, we modified cachegrind of valgrind tools [8], and then collected memory traces using it while running four applications; freecell, shotwell, gnuplot, and gqview. The characteristic of gathered memory traces is depicted as shown 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.

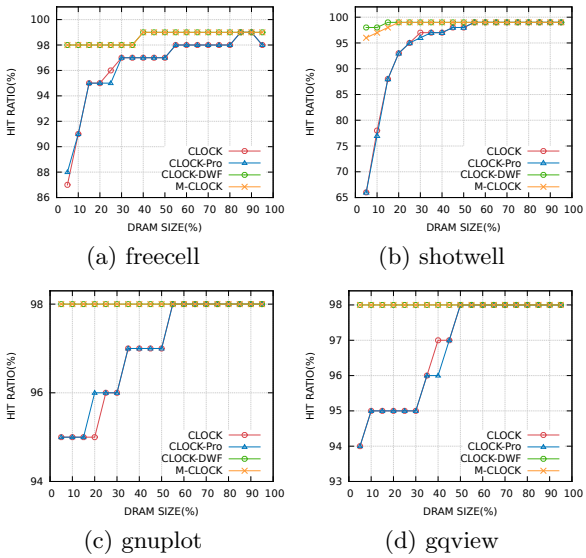
Workload	Memory footprint (KB)	Ratio of operations (Write : Read)
freecell	21,300	1 : 3.4
shotwell	42,728	1 : 1.5
gnuplot	4,540	4.1 : 1
gqview	58,964	1 : 1.3

Table 3: The characteristics of each workload

4.1 The Number of PCM Writes

In this subsection, we present the write-hit ratio in DRAM and the number of PCM writes, for each page replacement algorithm, because these are important factors in improving the performance of the *hybrid memory architecture*. Figure 2 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 2, M-CLOCK outperforms the other page replacement algorithms by up to 34%. It is because that 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.

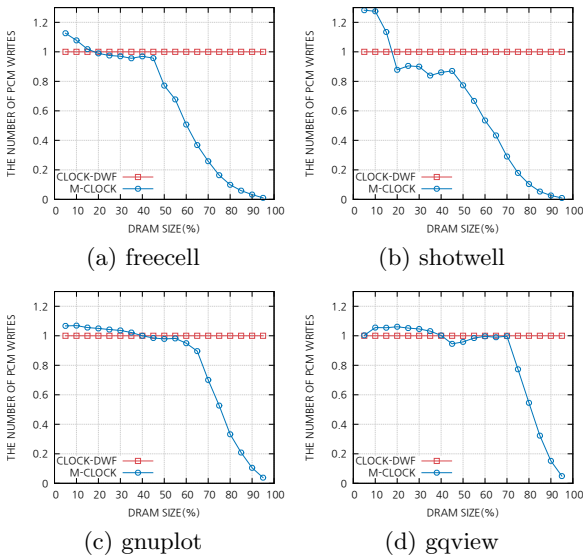
Figure 3 shows the number of PCM writes of M-CLOCK normalized to that of CLOCK-DWF. Since CLOCK and CLOCK-Pro have a large number of PCM writes in that they do not migrate the write-intensive pages from PCM to DRAM, they are excluded from the graph in Figure 3. The number of PCM writes of M-CLOCK is similar to that of CLOCK-DWF when the percentage of DRAM size is small in the *hybrid memory architecture*. However, as the percentage of DRAM size increases, M-CLOCK further reduces the number of PCM writes, by up to about 98% compared with



(a) freecell (b) shotwell

(c) gnuplot (d) gqview

Figure 2: The write-hit ratio in DRAM



(a) freecell (b) shotwell

(c) gnuplot (d) gqview

Figure 3: The number of PCM writes of M-CLOCK normalized to that of CLODK-DWF

CLOCK-DWF. In case of CLOCK-DWF, it does not fully use the capacity of DRAM for the read-intensive workload because it allocates most of the faulted pages on PCM according to the page access type when page fault occurs. On the other hand, 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 in DRAM and then migrates some pages from DRAM to PCM when no free page exists in DRAM.

4.2 The Amount of PCM Writes

Since a write operation is requested at various sizes of byte granularity, the amount of PCM writes differently affects the performance of *hybrid memory architecture* compared with the number of PCM writes. For this reason, we evaluate

the amount of PCM writes for each page replacement algorithm. Figure 4 shows that the amount of PCM writes of M-CLOCK normalized to that of CLOCK-DWF. As shown in Figure 4, M-CLOCK reduces the amount of PCM writes by up to 8% compared with CLOCK-DWF when the percentage of DRAM size is small in the *hybrid memory architecture*. Moreover, the gap in the amount of PCM writes becomes considerably large, by up to 99% when the percentage of DRAM size increases. It is because that M-CLOCK copes with the write operations in PCM according to *lazy migration*. In case of CLOCK-DWF, to reduce the number of PCM writes, it does not overwrite a requested 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, where a page migration from PCM to DRAM causes another page migration in reverse direction when no free page exists in both DRAM and PCM. On the other hand, M-CLOCK allows in-place update when write operation occurs on a page in PCM according to *lazy migration*, and then it migrates the page from PCM to DRAM when write operation occurs again. By doing so, M-CLOCK can reduce the migration-thrashing problem and have small average amount of PCM writes compared with CLOCK-DWF (i.e., 4KB write).

4.3 Effective Memory Access Time

To demonstrate the relation between PCM writes and the performance of *hybrid memory architecture*, we compare the effective memory access time of *hybrid memory architecture* for each page replacement algorithm (see Figure 5). The effective memory access time is described as Equation 1 with the parameters in Table 4.

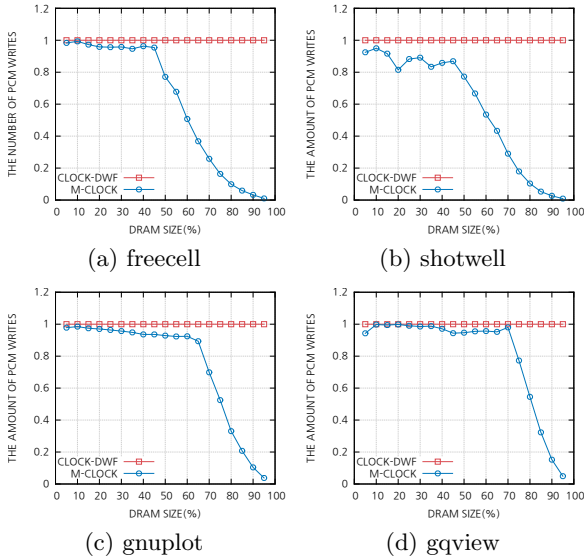
$$T_E = \alpha \times T_M + (1-\alpha) \times T_F$$
$$T_M = \beta \times T_P + (1-\beta) \times T_D \quad (1)$$
$$T_D = 50,\ T_P = (N_{P_r} \times L_{P_r} + N_{P_w} \times L_{P_w})/(N_{P_r} + N_{P_w})$$

Value	Description
T_E	Effective memory access time
T_M	Average memory access time
T_F	Page fault processing time
T_P, T_D	Average PCM or DRAM access time
α	The memory hit ratio
β	The probability that the page exists in PCM
N_{P_w}, N_{P_r}	The number of PCM writes/reads
L_{P_w}, L_{P_r}	The write/read latency of PCM

Table 4: The description of Equation 1

T_M refers to the average memory access time in *hybrid memory architecture* and T_F refers to the processing time caused by page fault. For the detail comparison in respect of memory access time, T_F is excluded in Figure 5. 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 (N_{P_r}) due to the asymmetric read/writs latency of PCM, of which the read latency (L_{P_r}) differ from its write latency (L_{P_w}) .

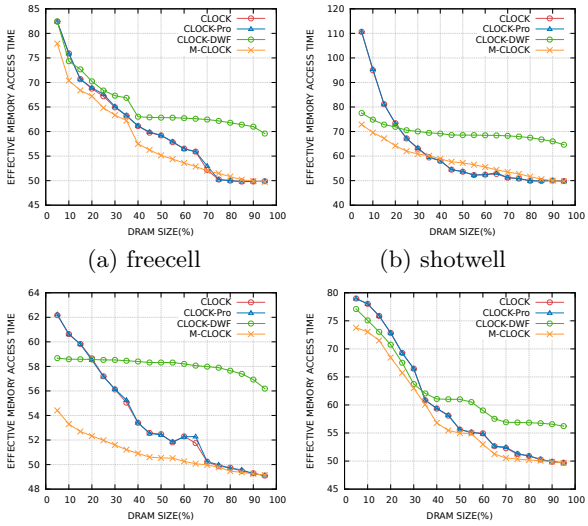
As shown in Figure 5, M-CLOCK has lower effective page access time by up to 34% compared with other page replacement algorithms. Since M-CLOCK well holds write-intensive pages in DRAM, T_E of M-CLOCK is relatively lower than that of CLOCK and CLOCK-Pro whose T_P is



(a) freecell (b) shotwell

(c) gnuplot (d) gqview

Figure 4: The amount of PCM writes of M-CLOCK normalized to that of CLODK-DWF



(a) freecell (b) shotwell

(c) gnuplot (d) gqview

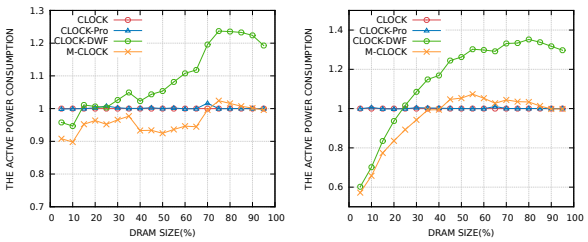
Figure 5: The effective memory access time

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 workload as aforementioned in Section 2.

4.4 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*.

$$P_t = P_s + P_a \quad (2)$$



(a) freecell (b) shotwell

(c) gnuplot (d) gqview

Figure 6: The active power consumption of M-CLOCK normalized to that of CLOCK

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

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 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 which 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 6 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 the other page replacement algorithms. It is because that M-CLOCK reduces the number of PCM writes by inducing most write operations to occur on DRAM as aforementioned.

5. CONCLUSIONS

In this paper, we propose a novel page replacement algorithm, called M-CLOCK, for *hybrid memory architecture* that consists of DRAM and PCM. M-CLOCK focuses on reducing the number of PCM writes, considering the weaknesses of PCM such as high write latency and short endurance. To achieve this, M-CLOCK keeps write-intensive pages in DRAM, 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. M-CLOCK also reduces the amount of PCM writes by using *lazy migration*, which delays migration (from PCM to DRAM) of the page requested by write operation, allowing to overwrite the data on the pages in PCM. Experimental results show that M-CLOCK reduces the number of PCM writes and improves the performance of *hybrid memory architecture*, by up to 98% and 34%, respectively.

6. ACKNOWLEDGMENTS

This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT & Future Planning (2010-0020730). Young Ik Eom is the corresponding author of this paper.

7. REFERENCES

[1] G. Dhiman, R. Ayoub, and T. Rosing. PDRAM: A Hybrid PRAM and DRAM Main Memory System. In *Proceedings of Design Automation Conference(DAC)*, pages 664–669. IEEE, 2009.

[2] S. Eilert, M. Leinwander, and G. Crisenza. Phase Change Memory: A New Memory Enables New Memory Usage Models. In *Proceedings of International Memory Workshop(IMW)*, pages 1–2. IEEE, 2009.

[3] A. P. Ferreira, M. Zhou, S. Bock, B. Childers, R. Melhem, and D. Mossé. Increasing PCM Main Memory Lifetime. In *Proceedings of Design, Automation and Test in Europe(DATE)*, pages 914–919. European Design and Automation Association, 2010.

[4] S. Jiang, F. Chen, and X. Zhang. CLOCK-Pro: An Effective Improvement of the CLOCK Replacement. In *Proceedings of USENIX Annual Technical Conference(ATC)*, pages 323–336, 2005.

[5] D. Kim, S. Lee, J. Chung, D. H. Kim, D. H. Woo, S. Yoo, and S. Lee. Hybrid DRAM/PRAM-based Main Memory for Single-chip CPU/GPU. In *Proceedings of Design Automation Conference(DAC)*, pages 888–896. ACM, 2012.

[6] B. C. Lee, P. Zhou, J. Yang, Y. Zhang, B. Zhao, E. Ipek, O. Mutlu, and D. Burger. Phase Change Technology and the Future of Main Memory. *IEEE Micro*, 30(1):131–141, 2010.

[7] S. Lee, H. Bahn, and S. Noh. CLOCK-DWF: A Write-History-Aware Page Replacement Algorithm for Hybrid PCM and DRAM Memory Architectures. *IEEE Transactions on Computers*, 63(9):2187–2200, 2013.

[8] N. Nethercote and J. Seward. Valgrind: A Program Supervision Framework. *Elsevier Electronic Notes in Theoretical Computer Science*, 89(2):44–66, 2003.

[9] M. K. Qureshi, V. Srinivasan, and J. A. Rivers. Scalable High Performance Main Memory System using Phase Change Memory Technology. *ACM SIGARCH Computer Architecture News*, 37(3):24–33, 2009.

[10] H. Seok, Y. Park, K.-W. Park, and K. H. Park. Efficient Page Caching Algorithm with Prediction and Migration for a Hybrid Main Memory. *ACM SIGAPP Applied Computing Review*, 11(4):38–48, 2011.

[11] A. S. Tanenbaum and Woodhull. *Operating Systems: Design and Implementation*. Prentice-Hall Englewood Cliffs, NJ, 1987.

[12] Wikipedia. Memory Refresh. `http://en.wikipedia.org/wiki/Memory_refresh/`, April 2014.

[13] P. Zhou, B. Zhao, J. Yang, and Y. Zhang. A Durable and Energy Efficient Main Memory using Phase Change Memory Technology. *ACM SIGARCH Computer Architecture News*, 37(3):14–23, 2009.