Page Replacement Algorithm with Lazy Migration for Hybrid PCM and DRAM Memory Architecture

Minho Lee, Dong Hyun Kang, Junghoon Kim, Young Ik Eom
Sungkyunkwan University, Korea
{minhozx, kkangsu, myhuni20, yieom}@skku.edu

1 Introduction

Recently, Phase Change Memory (PCM) has drawn great attention because it has many attractive features such as byte-addressable access, non-volatility, and in-place update. Especially, since PCM outperforms DRAM in terms of power consumption and scalability, it is promising alternative to DRAM for main memory of computer systems. Unfortunately, there are two limitations for whole main memory to be replaced with PCM: (1) each PCM cell supports limited number of write operations, only a million times, and (2) the write latency of PCM is expected to be slower than its read latency [1]. To solve these problems, several studies have focused on *hybrid memory architecture* [1, 3], which consists of DRAM and PCM, and proposed several page replacement algorithms for this architecture [2, 4].

Previous studies [2, 4] exploit the conventional page replacement algorithms (e.g., LRU, CLOCK), which are widely used for high hit ratio. To reduce the number of write operations on pages in PCM, they logically partition the main memory, which has fully associative *hybrid memory architecture*, into several regions and manage the regions with their own scheme. CLOCK-DWF [2] partitions its set of pages into two regions according to page characteristics such as major page access type (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 to DRAM when write operation occurs on the page in PCM. However, it has two critical problems. First, it generates unnecessary migration cost because it migrates a page from PCM to DRAM without considering whether DRAM is full or not. Second, it does not fully use total capacity of hybrid main memory in read or write-intensive workload because it only allocates faulted pages according to page access type when page fault occurs. Seok et al. [4] logically partitions the main memory into four regions: DRAM read, DRAM write, PRAM read, and PRAM write. It makes a decision on which pages should be migrated according to their *weighting value*. However, it leads to high performance overhead because the *weighting value* should be recalculated whenever the page is referenced.

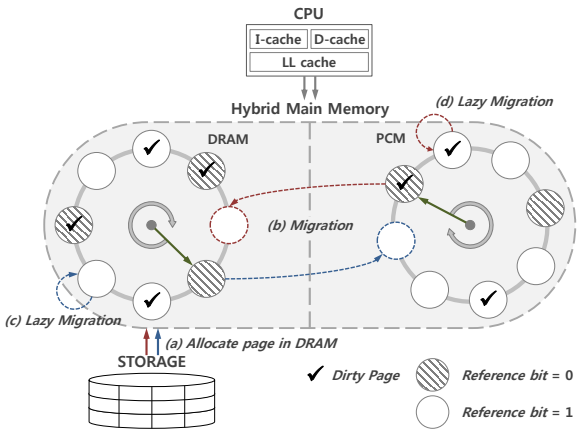

Figure 1: Overview of page replacement algorithm

In this paper, we propose a novel page replacement algorithm for *hybrid memory architecture* to reduce overhead of migration using *lazy migration*. Experimental results show that our scheme outperforms the previous work in terms of hit ratio and the number of write operations in PCM regardless of the PCM size in *hybrid memory architecture*.

2 Implementation with Lazy Migration

We propose a novel page replacement algorithm for *hybrid memory architecture*. Figure 1 shows the overview of our scheme based on the CLOCK algorithm. It effectively utilizes the total capacity of hybrid main memory by performing the page migration that maintains write-intensive pages in DRAM and read-intensive pages in PCM. Our scheme also reduces overhead of unnecessary migration using the *lazy migration*.

Page fault: When a page fault occurs, our scheme first loads the faulted page into DRAM as shown in Figure 1a, because DRAM outperforms PCM in terms of write latency and endurance. When DRAM has no free space for allocating the page, our scheme secures free space by migrating a page from DRAM to PCM. For this reason, our scheme fully utilizes the total capacity of hybrid main memory regardless of read or write-intensive workload.

Migration from DRAM to PCM: When there are

no free space in DRAM, our scheme scans the pages in DRAM to select a page whose reference bit is unset using a CLOCK-hand and then checks status of the selected page for page migration. If clean page is selected, our scheme immediately migrates it into PCM(Figure 1b). Otherwise, our scheme compares its *lazy count* (*LC*), which maintains how many times the migration of the page was delayed, with *migration threshold* for DRAM (*MT*$_{DRAM}$). If *LC* is smaller than *MT*$_{DRAM}$, our scheme increments its *LC* and moves the CLOCK-hand to the next page(Figure 1c). Otherwise, the selected page is migrated into PCM because it is considered as the most unlikely to be written. After migrating the page, *LC* of the page is reset for next *lazy migration*.

Migration from PCM to DRAM: When a write operation occurs on a page in PCM, our scheme tries to migrate the page into DRAM. If there are free space in DRAM, the page is migrated into DRAM and then updated by write operation(Figure 1b). Otherwise, our scheme compares its *LC* with *MT* for PCM (*MT*$_{PCM}$). If *LC* is smaller than *MT*$_{PCM}$, our scheme performs *lazy migration* that defers the page migration and just overwrites the data requested by the write operation and increments *LC* for reducing overhead of unnecessary migrations(Figure 1d). This is to prevent migration thrashing, or Ping-Pong effect, which is a situation that a migration causes another migration in a reverse direction. On the other hands, If *LC* is larger than *MT*$_{PCM}$, our scheme immediately migrates it into DRAM to avoid performance and lifetime degradation of PCM and then *LC* of the page is reset as above mentioned. When there are no free space in PCM, our scheme evicts a page according to basic rules of the CLOCK algorithm.

Unlike CLOCK-DWF, our scheme fully utilizes *hybrid memory* regardless of the characteristics of workloads by allocating faulted pages into DRAM and migrating pages from DRAM to PCM. Especially, when both DRAM and PCM have no free space, our scheme reduces migration overhead by using *lazy migration*.

3 Experimental Results

To varify our scheme, we implemented our prototype and the simulator for *hybrid memory architecture* that consists of DRAM and PCM. We also modified *cachegrind* to extract virtual address and page access type and collected them by running gedit. To compare our prototype with other replacement algorithms such as CLOCK and CLOCK-DWF, we measured the hit ratio and the number of write operations in PCM. In our evaluation, the size of total memory is set to the maximum memory usage on the workload. *MT*$_{DRAM}$ is set to 8 similarly to the overlooked rotation count in CLOCK-DWF, for comparison. *MT*$_{PCM}$ is set to 2 for giving

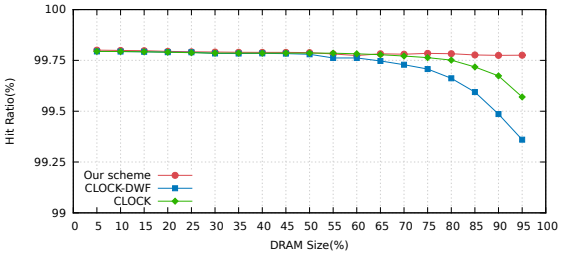

Figure 2: Comparison of hit ratio

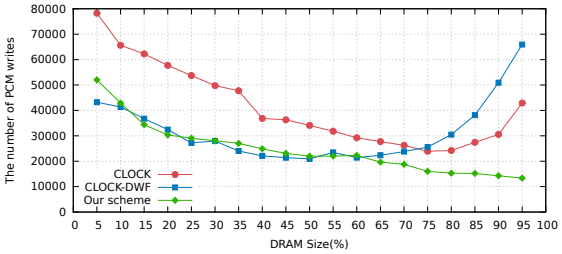

Figure 3: Comparison of the number of PCM writes

only a second chance. We evaluate various DRAM and PCM size. Experimental results show that our scheme presents high hit ratio in contrast with other algorithm regardless of PCM size in *hybrid memory architecture* as shown in Figure 2. In addition, as shown in Figure 3, our scheme reduces the number of PCM writes by up to 75% compared to CLOCK and CLOCK-DWF. This is because that our scheme takes advantage of in-place update in *lazy migration*.

Acknowledgements

This research was supported by the MSIP(Ministry of Science, ICT&Future Planning), Korea, under the ITRC(Information Technology Research Center) support program (NIPA-2013- H0301-13-4006) supervised by the NIPA(National IT Industry Promotion Agency)

References

[1] L. M. C. Eilert, Sean and Giuseppe. Phase change memory: A new memory enables new memory usage models. pages 1–2. IEEE, 2009.

[2] S. Lee, H. Bahn, and S. Noh. Clock-dwf: A write-history-aware page replacement algorithm for hybrid pcm and dram memory architectures. *IEEE Transaction on Computers*, 2013.

[3] 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.

[4] H. Seok, Y. Park, K.-W. Park, and K. H. Park. Efficient page caching algorithm with prediction and migration for a hybrid main memory. *SIGAPP Appl. Comput. Rev.*, 11(4):38–48, 2011.