A Buffer Replacement Algorithm for Providing Self-Logging in Hybrid Memory Architecture [*]

Dong Hyun Kang and Young Ik Eom
Sungkyunkwan University, South Korea
{kkangsu, yieom}@skku.edu

Motivation: High density non-volatile memory (NVM) is being considered as a potential successor to DRAM since it is a byte-addressable and persistent medium. Therefore, previous research that deploys NVM as main memory focused on how to gain the benefits of non-volatility of NVM. Most researchers introduced the hybrid memory architecture, which allows a page to be placed anywhere in DRAM or in NVM, in the way that reduces idle energy consumption and they proposed buffer replacement algorithms to compensate weaknesses of NVM, such as limited endurance and slower access latency (2-4x longer read and 7-50x longer write) compared to DRAM. However, proposed algorithms have a limited impact on the system performance due to the *latency issue* caused by the performance gap between DRAM and NVM. On the other hand, the primary focus of other researchers has been to revisit existing software stacks and mechanisms for improving overall system performance and a buffer replacement algorithm, called UBJ, is proposed, where it can guarantee both high performance and *reliability* by providing journaling functionalities via high performance NVM (e.g., spin transfer torque RAM). Unfortunately, for write-intensive workloads, UBJ may suffer from frequent copy-on-write (COW) because it duplicates a whole dirty page even though the page is modified just partially by small size update.

Our Solution: In this paper, we focus on the hybrid memory architecture comprised of DRAM and phase-change memory (PCM). We also propose a novel buffer replacement algorithm, called self-logging LRU (SL-LRU), that provides an opportunity to perform either page-granularity logging (PGL) or byte-granularity logging (BGL) within a buffer cache layer. In that way, SL-LRU not only improves overall performance but also guarantees *reliability* without storage accesses for journaling. Also, to avoid the waste of CPU cycles caused by *latency issue*, SL-LRU separately manages all pages in DRAM and PCM with two least recently used (LRU) lists: LRU_{DRAM} and LRU_{PCM}. When a page fault occurs with a read operation in demand paging system, SL-LRU firstly places the page on LRU_{DRAM} to take advantage of DRAM. On the other hand, when write fault occurs, SL-LRU places the faulted page on LRU_{PCM}. If LRU_{DRAM} is full, SL-LRU selects a page at the end of the LRU_{DRAM} as a victim page, and then the page is either migrated to LRU_{PCM} or evicted depending on its status. Whenever a free page is needed on LRU_{PCM}, SL-LRU prefers to evict clean pages over dirty pages and, among dirty pages, it prefers to evict *logged dirty pages* that are securely logged by the last *logging transaction* of self-logging.

Self-Logging: In self-logging, a commit operation is periodically triggered every 5 seconds, which is similar to the commit period for journaling in ext file systems. Also, when the total number of dirty pages on each LRU list reaches the maximum capacity of LRU_{PCM}, a commit is triggered to maintain all dirty pages in a *logging transaction*. If a small update, which modifies smaller area than half of a page, occurs on a *logged dirty page* in LRU_{PCM}, we first copy the part, which is to be modified in the page, to another page on LRU_{PCM} with its metadata, and then reflect the up-to-date data to the page at byte-level granularity. On the other hand, if a large update, that modifies larger area than half of a page, occurs on a *logged dirty page* in LRU_{PCM}, we copy the page on LRU_{PCM} to LRU_{DRAM} and the large update is reflected to the copied page on LRU_{DRAM} for PGL. When the copied page on LRU_{DRAM} is selected as a victim page of LRU_{DRAM}, we first copy it to LRU_{PCM} by allocating a free page. Finally, the last *logged dirty pages* and pages that store partial updates and its metadata of small updates are evicted from LRU_{PCM} through a commit operation.

Evaluation: We developed a trace-driven cache simulator and evaluated the cache hit ratio and performance with real mobile workloads. Our experimental results show that SL-LRU maintains comparable cache hit ratios to other algorithms, such as LRU, CLOCK, Linux2Q, and UBJ, even though it sometimes stores replicas on LRU_{DRAM} or LRU_{PCM} for self-logging. In addition, the simulated performance for read and write operations confirms that SL-LRU outperforms the other algorithms by up to 94%.

[*]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)