Efficient Hardware-Assisted Out-of-Place Update for Non-Volatile Memory*

Miao Cai†
Computer Science
Nanjing University

Chance C. Coats, Jeonghyun Woo, Jian Huang
Systems Platform Research Group, ECE Department
University of Illinois at Urbana-Champaign

***Abstract*—Byte-addressable non-volatile memory (NVM) is a promising technology that provides near-DRAM performance with scalable memory capacity. However, it requires atomic data durability to ensure memory persistency. Therefore, many techniques, including logging and shadow paging, have been proposed. However, most of them either introduce extra write traffic to NVM or suffer from significant performance overhead on the critical path of program execution, or even both.**

In this paper, we propose a transparent and efficient hardware-assisted out-of-place update (HOOP) mechanism that supports atomic data durability, without incurring much extra writes and performance overhead. The key idea is to write the updated data to a new place in NVM, while retaining the old data until the updated data becomes durable. To support this, we develop a lightweight indirection layer in the memory controller to enable efficient address translation and adaptive garbage collection for NVM. We evaluate HOOP with a variety of popular data structures and data-intensive applications, including key-value stores and databases. Our evaluation shows that HOOP achieves low critical-path latency with small write amplification, which is close to that of a native system without persistence support. Compared with state-of-the-art crash-consistency techniques, it improves application performance by up to 1.7×, while reducing the write amplification by up to 2.1×. HOOP also demonstrates scalable data recovery capability on multi-core systems.

I. Background and Motivation

Non-volatile memory (NVM) like PCM, STT-RAM, ReRAM, and 3D XPoint offers promising properties, such as byte-addressability, non-volatility, and scalable capacity. Unlike DRAM-based systems, applications using NVM require memory persistency to ensure crash safety, which means a set of data updates must behave in an atomic, consistent, and durable manner, with respect to system failures and crashes.

Ensuring memory persistency with commodity out-of-order processors and hardware-controlled cache hierarchies, however, is challenging and costly due to unpredictable cache evictions. Prior researches have developed various crash-consistency techniques for NVM, such as logging, shadow paging, and their optimized versions. However, they either introduce extra write traffic to NVM, or suffer from significant performance overhead on the critical path of program execution, or even both.

Specifically, although logging provides strong atomic durability against system crashes, it introduces significant overheads (see Table I). First, both undo logging and redo logging must make a data copy before performing the in-place update. Persisting these data copies incurs extra writes to NVM on the critical path of program execution. This not only decreases application performance, but also hurts NVM lifetime. Second, enforcing the correct persistence ordering between log and data updates requires cache flushes and memory fences, which further causes significant performance overheads.

To address the aforementioned problems, researchers recently proposed asynchronous in-place updates, in which the systems maintain an explicit main copy of data to perform in-place updates, and then asynchronously apply these changes to the data copy, or asynchronously persist the undo logs to NVM. Unfortunately, it does not mitigate the problem of incurring additional write traffic, due to the background data synchronization. An alternative technique, shadow paging, incurs both additional data writes to NVM and performance overhead on the critical path, due to its copy-on-write (CoW) mechanism, as shown in Table I. Although an optimized shadow paging method that enables data copies at cache-line granularity was proposed, it requires TLB modifications to support the cache-line remapping. Another approach is the software-based log-structured memory, which reduces the persistency overhead by appending all updates to logs. However, it requires multiple memory accesses to identify the data location for each read, which incurs significant critical-path latency.

II. Design and Implementation

In this paper, we propose a transparent hardware-assisted out-of-place (OOP) update approach, named HOOP. The key idea of HOOP is to store the updated data outside of their original locations in dedicated memory regions in NVM, and then apply these updates lazily through an efficient garbage collection scheme. HOOP reduces data persistence overheads in three aspects. First, it eliminates the extra writes caused by the logging mechanisms, as the old data copies already exist in NVM and logging is not required. Second, the out-of-place update does not assume any persistence ordering for store operations, which allows them to execute in a conventional out-of-order manner. Third, persisting the new updates in new locations does not affect the old data version, which inherently supports the atomic data durability.

Since the update is written to a new place in NVM, we develop a lightweight indirection layer in the memory controller to handle the physical address remapping. HOOP

*This work has been published at ISCA'20 [1].
†Work done while visiting the Systems Platform Research Group at UIUC.

TABLE I: Comparison of various crash-consistency techniques for NVM. Compared with existing works, HOOP provides a transparent hardware solution that significantly reduces the write traffic to NVM, while achieving low persistence overhead.

Approach	Subtype	Read Latency	On the Critical Path	Require Flush & Fence	Write Traffic
Logging	Undo	Low	Yes	No	Medium/High
	Redo	High	Yes	Yes	Medium/High
	Undo+Redo	High	Yes	No	High
Shadow paging	Page	Low	Yes	Yes	High
	Cache line	Low	Yes	Yes	Low
Log-structured NVM		High	No	No	Medium
HOOP		**Low**	**No**	**No**	**Low**

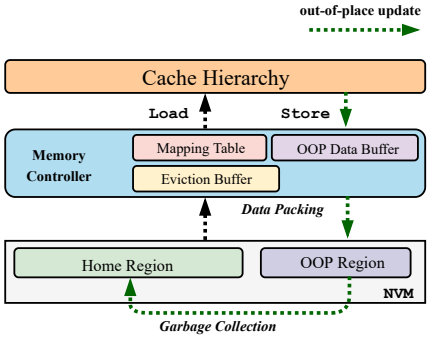

Fig. 1: HOOP performs out-of-place writes and reduces write traffic with *data packing and coalescing*. To reduce the storage overhead, HOOP adaptively migrates data in the out-of-place (OOP) region back to the home region with optimized GC.

enables high-performance and low-cost out-of-place update with four major components. First, we organize the dedicated memory regions for storing data updates in a log-structure manner, and apply data packing to the out-of-place updates. This makes HOOP best utilize the memory bandwidth of NVM as well as reduce the write traffic to NVM. Second, to reduce the memory space cost caused by the out-of-place updates, HOOP develops an efficient garbage collection (GC) algorithm to adaptively restore the out-of-place updates back to their home locations. To further reduce the data movement overhead during GC, we exploit a data coalescing scheme that combines the updates to the same cache lines. Therefore, HOOP only need to restore multiple data updates once, which further reduces the additional write traffic. Third, HOOP maintains a hash-based address-mapping table in HOOP for physical-to-physical address translation, and ensures that load operations always read the updated data from NVM with trivial address translation overhead. Since the entries in the address-mapping table will be cleaned when the corresponding out-of-place updates are periodically written back to their home addresses, the mapping table size is small. Fourth, HOOP enables fast data recovery by leveraging the thread parallelism available in multi-core computing systems.

We present the architectural overview of HOOP in Figure 1. During transaction execution, data is brought into the cache hierarchy with `load` and `store` operations. They will access the indirection layer to find the most recent version of the desired cache line. For the updated cache lines in a transaction, they are buffered in the *OOP data buffer* in HOOP. Each entry of this buffer can hold multiple data updates as well as the associated metadata. And persistence optimizations such as data packing are applied to improve the transaction performance, when flushing the updated cache lines to the OOP region. With the out-of-place writes, HOOP is crash-safe by ensuring committed transactions are persisted in the OOP region before any changes are made to the original addresses (i.e., the home region). As OOP region will be filled with updated data and metadata, HOOP performs periodic GC to migrate the most recent data versions to the home region, and uses *data coalescing* to minimize the write traffic to NVM. Upon power failures or system crashes, HOOP will leverage thread parallelism to scan the OOP region and instantly recover the data to a consistent state.

As HOOP is developed in the memory controller, it is transparent to upper-level systems software. No non-volatile cache or TLB modifications for address translation are required. Unlike software-based logging approaches that suffer from long critical-path latency for read operations, HOOP provides an efficient hardware solution with low performance overhead and write traffic. In summary, we make the following contributions in this paper:

- We present a hardware out-of-place update scheme to ensure the crash-consistency for NVM, which alleviates extra write traffic and avoids critical-path latency overhead in NVM.
- We propose a lightweight persistence indirection layer in the memory controller with minimal hardware cost, which makes out-of-place updates transparent to software systems.
- We present an efficient and adaptive GC scheme, which will apply the recent data updates from the out-of-place update memory regions to their original locations for memory space saving and write traffic reduction.

III. EVALUATION

We developed HOOP in a Pin-based many-core simulator, McSimA+, with the combination of an NVM simulator. We evaluated HOOP against four representative crash-consistency approaches, including undo logging, redo logging, optimized shadow paging, and log-structured NVM. We used a set of microbenchmarks running against these popular data structures like hashmaps, and real-world application workloads like Yahoo Cloud Service Benchmark (YCSB) and transactional databases. Experimental results demonstrate that HOOP significantly outperforms state-of-the-art approaches, while ensuring the same atomic durability as existing crash-consistency techniques. We presented the detailed evaluation in [1].

REFERENCES

[1] M. Cai, C. C. Coats, and J. Huang, "Hoop: efficient hardware-assisted out-of-place update for non-volatile memory," in *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA'20)*, Virtual Event, 2020. [Online]. Available: http://jianh.web.engr.illinois.edu/papers/hoop-isca2020.pdf