Asynchronous Persistence with ASAP*

Ahmed Abulila
Microsoft Corporation

Izzat El Hajj
American University of Beirut

Myoungsoo Jung
Korea Advanced Institute of Science and Technology

Nam Sung Kim
University of Illinois at Urbana-Champaign

1 MOTIVATION

The problem our work tackles is hardware logging for persistent memory. It is important because it promises higher performance than pure software solutions.

Persistent memory provides the byte-addressability and low latency of DRAM as well as the persistency of storage devices. Programming persistent data structures typically involves writing atomic regions that are atomic and durable, ensuring that stores to persistent memory within an atomic region persist in an all-or-none manner. To guarantee the atomicity and durability of atomic regions, Write-Ahead Logging (WAL) has commonly been used [2]. WAL consists of two key persist operations: log persist (LPO) and data persist (DPO) [5]. LPO operations flush log entries to persistent memory before making the data persistent, to ensure that a consistent state can be recovered if a crash occurs before all the data of a given atomic region has persisted. DPO operations write back the actual data modified by the atomic region to persistent memory.

Software logging solutions [8] for persistent memory offload the complexity of managing logs to the software, placing persist operations on the critical path of an atomic region's execution. In contrast, hardware logging solutions [4, 7] for persistent memory can initiate persist operations in a manner that is transparent to software, and can complete these operations in the background, overlapping them with the execution of other instructions and ultimately resulting in better performance.

2 LIMITATIONS OF THE STATE OF THE ART

The key limitation of state-of-the-art hardware solutions is that hardware *undo* logging solutions do not support asynchronous persist operations, whereas hardware *redo* logging solutions support asynchronous data persist operations but not asynchronous log persist operations.

Persist operations that are overlapped with execution of other instructions can be classified as *synchronous* or *asynchronous* with respect to the end of an atomic region. Synchronous persist operations are overlapped with execution of other instructions *within* an atomic region, but once the end of an atomic region is reached, all persist operations initiated by the atomic region must complete before instruction execution proceeds. In contrast, asynchronous persist operations are also overlapped with instructions that are executed *after* the atomic region, hence allowing instruction execution to proceed past the end of an atomic region without waiting for persist operations to complete.

The advantage of asynchronous persist operations is that they reduce idle time by not waiting at the end of an atomic region. Moreover, in a multi-threaded context, if the atomic region is nested inside a critical section to guarantee isolation, the latency of asynchronous persist operations will not be part of the critical section [3]. Removing the persistence latency from critical sections benefits concurrency by reducing the execution time of critical sections.

The two major approaches to hardware WAL are undo-logging [7] and redo-logging [4]. While undo logging has several advantages over redo logging which are discussed in the paper, a key disadvantage is that hardware solutions for undo logging do not support asynchronous DPO operations, whereas hardware solutions for redo logging do. Neither support asynchronous LPO operations.

3 KEY INSIGHTS

The key insight in the paper is that asynchronous log and data persist operations can both be supported under undo logging if control and data dependences between atomic regions are tracked and enforced.

The challenge with supporting asynchronous persist operations under undo logging is that allowing instruction execution to proceed past the end of an atomic region before the atomic region has completed its persist operations may result in violating control and data dependencies between atomic regions. In other words, it runs the risk of a later atomic region committing before an earlier one does, or, in a multi-threaded context, a consumer atomic region committing before the corresponding producer atomic region does [6]. These situations leave the data in an unrecoverable state after a crash. The paper includes examples that demonstrate these situations.

We show that asynchronous LPO and asynchronous DPO operations can both be supported under undo logging by tracking control and data dependences between atomic regions and enforcing atomic regions commit (undo logs are freed) in a manner that respects these dependences.

4 MAIN ARTIFACTS

The main artifact presented in the paper is a hardware design that supports undo logging with asynchronous log and data persist operations by tracking and enforcing control and data dependencies between atomic regions in hardware.

The paper presents *ASAP*, the first **A**rchitecture **S**upport for **A**synchronous **P**ersistence (*ASAP*). *ASAP* allows both LPO and DPO operations to happen asynchronously, and tracks control and data dependencies between atomic regions in hardware to ensure that atomic regions commit (undo logs are freed) in the proper order. The hardware extensions that *ASAP* requires are shown in Figure 1. The key extensions include:

- Per-thread registers (❶) assist with log management

*This work has been published at ISCA '22 [1].

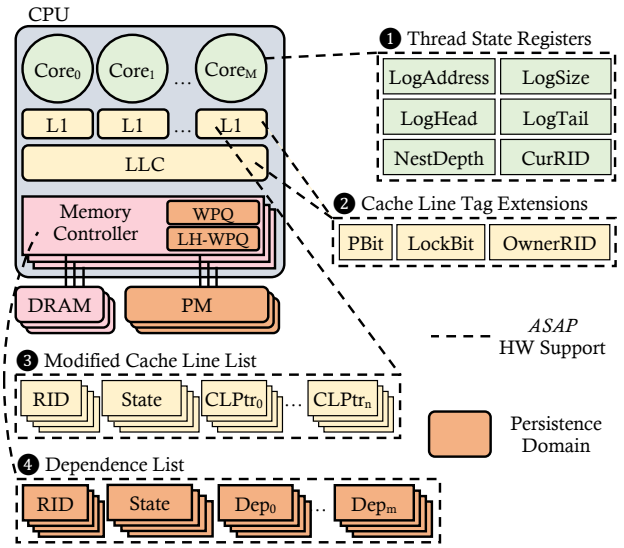

Figure 1: *ASAP* hardware extensions

- Structures to track which atomic region last modified a persistent cache line to help capture data dependecies between atomic regions (❷)
- Structures to track which cache lines have been modified by an atomic region to ensure that all the cache lines persist before the atomic region commits (❸)
- Structures to track which atomic regions are still active and the atomic regions they depend on to ensure that all an atomic region's dependencies have been resolved before its log is freed (❹)

ASAP provides a simpler software interface than prior hardware undo logging approaches. Also, *ASAP* applies three key optimizations to reduce persistent memory traffic: LPO dropping, DPO coalescing, and DPO dropping. These optimizations are particularly effective in combination with asynchronous persist operations in reducing memory traffic. *ASAP* is implemented and evaluated using gem5.

5 KEY RESULTS AND CONTRIBUTIONS

We show that *ASAP* improves performance compared to state-of-the-art hardware logging approaches that perform some persist operations synchronously.

We compare *ASAP* to the following baselines:

- **SW:** Software-only implementation of undo-logging
- **HWUndo:** Based on the state-of-the-art hardware undo logging implementation [7] which performs LPO and DPO operations synchronously
- **HWRedo:** Based on the state-of-the-art hardware redo logging implementation [4] which performs LPO operations synchronously
- **NP:** Data is read from and written to persistent memory, but no crash consistency is guaranteed

The performance improvement for select benchmarks is shown in Figure 2. More benchmarks are evaluated in the paper. *ASAP* improves performance compared to state-of-the-art hardware undo

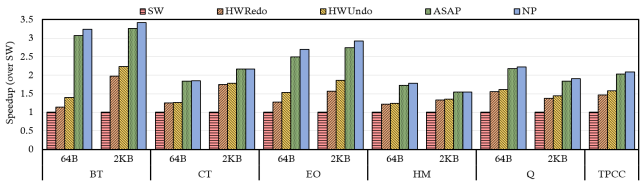

Figure 2: Performance comparison (speedup over SW) for select benchmarks

logging by 1.41× (geomean), while achieving 0.96× (geomean) of the ideal performance when no persistence is enforced. *ASAP* also reduces persistent memory traffic, generating 0.52× (geomean) the traffic generated by state-of-the-art hardware undo logging techniques. *ASAP* is robust against increasing persistent memory latency, and is therefore suitable for both fast and slow persistent memory technologies.

6 CONCLUSIONS

This paper presents *ASAP*, a hardware logging scheme that allows atomic regions to commit asynchronously. Committing atomic regions asynchronously removes the need to wait for log persist and/or data persist operations at the end of atomic regions, which reduces the latency of these regions. To ensure that the atomic regions commit in the proper order, *ASAP* tracks and enforces control and data dependencies between atomic regions in hardware. Our evaluation shows that *ASAP* outperforms state-of-the-art hardware undo and redo logging techniques, which commit atomic regions synchronously. It also reduces persistent memory traffic and is suitable for both fast and slow persistent memory technologies.

REFERENCES

[1] A. Abulila, I. E. Hajj, M. Jung, and N. S. Kim, "ASAP: Architecture support for asynchronous persistence," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ser. ISCA '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 306–319. [Online]. Available: https://doi.org/10.1145/3470496.3527399

[2] Q. Hu, J. Ren, A. Badam, J. Shu, and T. Moscibroda, "Log-Structured Non-Volatile Main Memory," in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. Santa Clara, CA: USENIX Association, 2017, pp. 703–717. [Online]. Available: https://www.usenix.org/conference/atc17/technical-sessions/presentation/hu

[3] J. Jeong and C. Jung, "Pmem-spec: Persistent memory speculation (strict persistency can trump relaxed persistency)," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 517–529. [Online]. Available: https://doi.org/10.1145/3445814.3446698

[4] J. Jeong, C. H. Park, J. Huh, and S. R. Maeng, "Efficient Hardware-Assisted Logging with Asynchronous and Direct-Update for Persistent Memory," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2018, pp. 520–532.

[5] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwarz, "ARIES: A Transaction Recovery Method Supporting Fine-granularity Locking and Partial Rollbacks Using Write-ahead Logging," *ACM Trans. Database Syst.*, vol. 17, no. 1, pp. 94–162, Mar. 1992.

[6] H. E. Ramadan, C. J. Rossbach, and E. Witchel, "Dependence-aware transactional memory for increased concurrency," in *2008 41st IEEE/ACM International Symposium on Microarchitecture*, 2008, pp. 246–257.

[7] S. Shin, S. K. Tirukkovalluri, J. Tuck, and Y. Solihin, "Proteus: A Flexible and Fast Software Supported Hardware Logging Approach for NVM," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-50 '17. New York, NY, USA: ACM, 2017, pp. 178–190. [Online]. Available: http://doi.acm.org/10.1145/3123939.3124539

[8] H. Volos, A. J. Tack, and M. M. Swift, "Mnemosyne: Lightweight Persistent Memory," in *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XVI, Newport Beach, CA, 2011, pp. 91–104.