Unbounded Hardware Transactional Memory for a Hybrid DRAM/NVM Memory System

Jungi Jeong*§, Jaewan Hong†, Seungryoul Maeng†, Changhee Jung*, Youngjin Kwon†
*Department of Computer Science, Purdue University
†School of Computing, KAIST

I. Introduction

Emerging Non-Volatile Memory (NVM) has gained massive attention from researchers and practitioners for the last decade. However, the benefits of NVM do not come for free. First, in-memory states of NVM must remain consistent across software crashes and machine failures, making *failure-atomicity* of NVM updates a hard requirement. Second, in-memory states must remain consistent even when they are concurrently updated. With that in mind, researchers have proposed `durable transactions` that support both atomic-durable updates and isolation in software. Since these proposals incur significant performance overheads, others take advantage of hardware support to accelerate a durable transaction. For example, they equip atomic-durable updates with hardware-based logging [2], [4], [6], [7] and offer isolation using hardware transactional memory (HTM) [1], [5].

However, the hardware support limits the total size of transactions, raising practical concerns. When a transaction goes over the hardware limitation (i.e., `capacity overflows`), it aborts and restarts, which significantly degrades the performance and the forward progress due to repeated overflows. Given that unbounded HTMs can address the capacity overflow problem, it would be worth adapting them to persistent programming.

Unfortunately, merely extending the state-of-the-art unbounded HTMs is not sufficient to support full-fledged persistent programming in practice. Previous unbounded HTMs either rely on unrealistic assumptions or suffer from an unacceptably large amount of false conflicts. More importantly, no prior study supports the interaction between non-persistent and persistent objects within a transaction. Since pointers between DRAM and NVM exhibit non-trivial inter-dependencies in the applications, their HTM acceleration must deal with the dependencies to provide consistency between persistent and non-persistent objects.

To overcome these challenges, we present UHTM, our hardware support for **U**nboundedness in the DRAM/NVM **H**ybrid **T**ransactional **M**emory. To efficiently isolate transactions, UHTM introduces `staged conflict detection` that combines two detection schemes and selectively uses one depending on the transactional object location. UHTM leverages the *cache-coherence protocol* for objects in on-chip caches that renders conflict detection accurate but limits the transactions within the on-chip cache boundary. On the other hand, when transactional objects are evicted to off-chip memory, UHTM uses *address signatures* based on hardware bloom filters that can cover the unlimited address space.

Moreover, UHTM further reduces the abort rate in virtualized or containerized environments. Even if two persistent applications do not share data, LLC miss requests of one can be found in the address signature of another. This false positive causes a transaction to abort. UHTM prevents such unnecessary abortion between different persistent applications by confining the conflict domain. Isolating transactions per conflict domain achieves an additional reduction on the false positive rate, making UHTM practical in consolidated environments.

To minimize the commit latency, which is common and performance-critical, UHTM proposes `hybrid logging` that brings two different logging techniques into the design. It performs *undo logging* for DRAM objects while *redo logging* for NVM objects.

II. UHTM

Atomic-Durability: Atomicity applies for both DRAM and NVM data and requires to make modifications visible at once when commits. UHTM supports atomicity via logging for both DRAM and NVM data. On the other hand, durability only correlates with NVM data. It requires to flush all changes to NVM before a transaction commits. Based on this observation, UHTM proposes the `hybrid` scheme that uses the `eager` version management for data in on-chip caches and DRAM while applying the `lazy` policy for NVM data. *All* NVM updates within transactions are logged with new value and flushed to NVM for durability. On the other hand, for DRAM data, logging is only used for *overflowed* blocks.

Consistency and Data Recovery: UHTM restores the program state from a power failure with NVM data only. UHTM replays the committed redo entries in the NVM log area and disregards the uncommitted ones, like the recovery of redo logging in the conventional database logging. The programmers' responsibility is to place data structures in NVM if they are necessary for data recovery. On the other hand, UHTM guarantees the consistency of both DRAM and NVM data in conflicted transactions. When a conflict is detected, UHTM aborts the conflicted transaction by rolling back to the previous consistent state of DRAM and NVM.

Isolation: UHTM guarantees serializability, where the concurrent and serial executions produce the same output. This guar-

§This work is mostly done when the author was a Ph.D. student at KAIST.

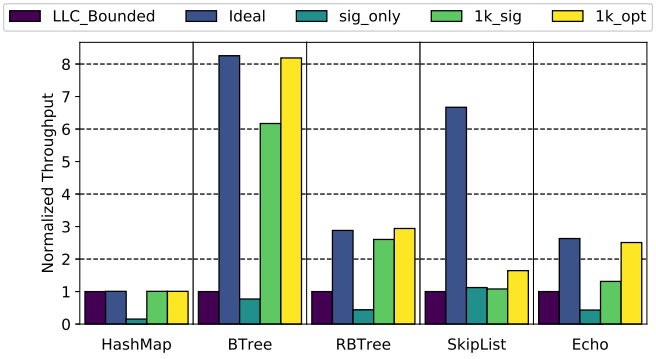

Fig. 1: Throughput of PMDK benchmarks and Echo key-value store running transactions of 100KB footprints. Plots are normalized to LLC-Bounded HTM.

antee is identical to what other hardware transactional memory systems guarantee both in commercial [3] and literature [1], [5]. UHTM proposes the `staged` conflict detection mechanism that uses different methods in on-chip caches and off-chip memory. UHTM extends the directory-based cache coherence protocol to detect conflicts in on-chip caches, which is accurate. At the same time, it uses address signatures for overflowed blocks in off-chip memory, which provide an unlimited range. UHTM uses the `eager` conflict detection policy for both on- and off-chip memory.

Optimization: Using address signatures for cache-overflowed blocks arises a unique challenge that aborts transactions if they conflict with a *non-transactional* context. Signature arrays are vulnerable to such false-conflicts since all LLC-missed requests must be checked against all signatures to provide correctness. For example, background processes, which are *completely unrelated* to durable transactions, could abort the transaction if signature arrays detect it as a conflict (but it is false-positive). To reduce the false-positive rates, UHTM proposes the signature isolation technique that confines address signatures by defining the conflict domain. The conflict domain denotes a group of transactions that share the address space and, therefore, potentially conflict with each other. We modified the pthread library to generate a transaction group ID shared by threads in the process. The signature isolation technique does not require modifications to applications.

III. Evaluation

Our evaluation uses the system-call emulation mode of the gem5 simulator and compares the following designs.

- **LLC-Bounded HTM:** limits the transaction boundary to on-chip caches, similar to [5].
- **Signature-Only HTM:** detects conflicts with only 1k-bit address signatures by checking all coherence traffics.
- **UHTM:** detects conflicts with the staged scheme (cache-coherence and 1k-bit signatures) and provides hybrid logging.
- **Ideal Unbounded HTM:** detects conflicts without producing a false-positive.

Figure 1 shows the throughput of PMDK benchmarks such as `HashMap`, `B-Tree`, `RB-Tree`, `SkipList`, and the real-world application, `KV-Echo`. Each benchmark either inserts or updates the persistent data structure with the value size of 100KB. We also run two memory-intensive applications to emulate contention in LLC.

First, the capacity overflow significantly hurts the throughput of durable transactions. `HashMap` does not experience the capacity overflows, Hence, the LLC-bounded HTM and UHTM do not show the difference in the transaction throughput. Nevertheless, the capacity overflow severely degrades the throughput. For example, `B-Tree` and `SkipList` benchmarks incur 8.2x and 6.7x slowdown while `RB-Tree` and `Echo` show 2.7x and 2.5x throughput degradation, respectively. In particular, UHTM offers a substantial speed-up for `B-Tree`, `RB-Tree`, and `Echo` benchmarks, making them comparable to the ideal performance. On the other hand, UHTM under-performs in `SkipList`, showing less than 2x speed-up compared to the LLC-Bounded baseline while the ideal one achieves 7x speed-up. It turns out that UHTM ends up with many false-positives since `SkipList` traverses the list, which significantly diminishes throughput by false conflicts.

Second, when consolidating multiple persistent applications, UHTM's optimization (noted `1k_opt` in Figure 1) that confines conflict domains of hardware transactions eliminates false aborts between different conflict domains. By grouping address signatures with conflict domains, UHTM substantially improves throughput since applications less abort and restart. Furthermore, the signature isolation technique removes false conflicts with memory-intensive applications. Hence, confining conflict domains is essential to reduce false-positives of signature-based approaches.

IV. Conclusion

We presented UHTM that supports unbounded hardware transactions for DRAM and NVM hybrid memory systems. UHTM leverages two techniques, i.e., the staged conflict detection and hybrid hardware logging schemes. Experimental results show that UHTM significantly outperforms the state-of-the-art that limits the transaction to the size of on-chip caches and supports NVM data only therein.

References

[1] D. Castro, P. Romano, and J. Barreto, "Hardware transactional memory meets memory persistency," in *IPDPS*, 2018.
[2] K. Doshi, E. Giles, and P. Varman, "Atomic persistence for scm with a non-intrusive backend controller," in *HPCA*, 2016.
[3] Intel, "Programming with intel® transactional synchronization extensions," *Intel® 64 and IA-32 Architectures Software Developer's Manual*, 2019.
[4] J. Jeong, C. H. Park, J. Huh, and S. Maeng, "Efficient hardware-assisted logging with asynchronous and direct-update for persistent memory," in *MICRO*, 2018.
[5] A. Joshi, V. Nagarajan, M. Cintra, and S. Viglas, "Dhtm: Durable hardware transactional memory," in *ISCA*, 2018.
[6] A. Joshi, V. Nagarajan, S. Viglas, and M. Cintra, "Atom: Atomic durability in non-volatile memory through hardware logging," in *HPCA*, 2017.
[7] M. A. Ogleari, E. L. Miller, and J. Zhao, "Steal but no force: Efficient hardware undo+redo logging for persistent memory systems," in *HPCA*, 2018.