Hardware support for ACID Transactions in Persistent Memory

Arpit Joshi *
Intel
arpit.joshi@intel.com

Vijay Nagarajan
University of Edinburgh
vijay.nagarajan@ed.ac.uk

Marcelo Cintra
Intel
marcelo.cintra@intel.com

Stratis Viglas
Google
sviglas@google.com

I. Introduction

The emergence of byte-addressable non-volatile memory technologies, also known as *persistent memory*, is fast blurring the divide between memory and storage. Being directly attached to the memory bus, persistent memory provides a high-bandwidth and low-latency alternative for durability. However, programmers need guarantees about what will remain in persistent memory upon a crash or a failure.

System support for systematic programming models such as transactions that provide ACID guarantees (Atomicity, Consistency, Isolation and Durability) is essential for writing crash consistent programs. ACID essentially implies that updates within a transaction are made visible (to other transactions) as well as durable (to a non-volatile medium), in an atomic manner. While the database community has developed a plethora of techniques to guarantee ACID efficiently, these techniques have predominantly been developed with slow block-based media in mind. When applied to in-memory settings, such techniques tend to spend a significant amount of time on concurrency control and logging (for durability). This leads us to ask the question: How fast can we enforce ACID in the presence of fast persistent memory?

Recently, there have been multiple proposals for providing ACID updates to persistent memory. These proposals either support both atomic visibility and atomic durability in software or support only atomic durability in hardware. Software support for atomic visibility or durability comes at a significant performance cost because of limited concurrency and a high overhead of durability [1], [2]. Moreover, performance improvement of hardware support for atomic durability is limited if atomic visibility is implemented in software. Additionally, certain proposals leverage commercially available Hardware Transactional Memory (HTM) to support atomic visibility in hardware. HTMs primarily provide support for three functionalities: buffering the speculative state, tracking read and write sets and detecting conflicts. Commercial HTMs typically buffer speculative state in private caches (typically L1). As a result, they efficiently support only small transactions [2]. Additionally, HTM systems only provide ACI guarantees, i.e., atomic visibility but not atomic durability. To guarantee ACID, HTMs are integrated with software support for atomic durability which increase the transaction's write set size (and consequently the abort rate).

Our aim is to design an HTM that can support ACID transactions efficiently while extending the supported transaction size beyond the L1 cache with minor changes to the coherence protocol or the caches. One approach is to leverage existing unbounded HTM designs that rely on logging to support overflows and make those logs durable. However, such an approach, where durability is treated as a secondary consideration, will have poor performance as persisting the log and/or the data will always be in the critical path. Additionally, these protocols have to stall requests when they conflict on cache lines that have overflowed from caches. Stalling adds significant design complexity as it requires support for retrying requests using a NACK based coherence protocol.

We advocate an alternative approach in which durability is a first class design constraint. We propose Durable Hardware Transactional Memory (DHTM) in which we integrate a commercial HTM with hardware support for redo logging. DHTM achieves atomic visibility by leveraging HTM. Whereas for achieving durability, DHTM provides architectural support for transparently and efficiently writing redo log entries to a durable transaction log maintained in persistent memory. DHTM additionally extends the supported transaction size by leveraging the same logging infrastructure for also supporting L1 overflows.

II. DHTM Design

Hardware Redo Log. We choose a redo-log based design as it allows us to have both fast commits as well as fast aborts for durable transactions. Redo logging enables fast commits as it requires only the log to be written to persistent memory at commit time, while allowing for data updates (write set) to be written back in the background. Redo logging also enables fast aborts consisting of two simple steps: invalidate the modified lines in the cache and discard the redo log. One drawback of redo logging is that, because a redo log entry needs to be created for every store, a large number of log entries need to be written to persistent memory which can consume significant amounts of memory write bandwidth. Our proposed hardware based redo-logging mechanism overcomes this limitation by allowing log entries modifying the same cache-line to be coalesced in the cache

* This work was performed while the author was at The University of Edinburgh.

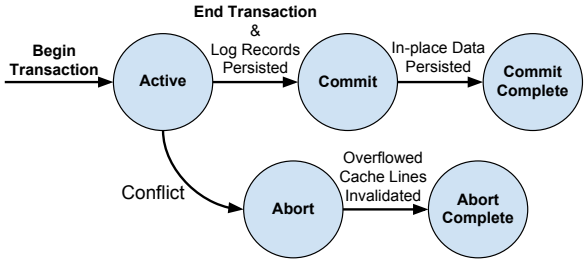

Figure 1: States of a transaction.

and subsequently writing line-granular log entries to persistent memory [3].

ACID Transactions. DHTM supports ACID transactions by combining hardware support for redo logging with HTMs. An ACID transaction in DHTM can be expressed in the form of a state diagram as shown in Figure 1. At the start of a new transaction, DHTM transitions to *Active* state. In the *Active* state, DHTM writes line granular redo log entries to persistent memory, for all the cache lines being modified in the transaction. Subsequently, the transaction can either take the commit path or the abort path, which are described below.

Commit: Upon reaching the end of the transaction, and having written all redo-log entries to persistent memory, DHTM transitions to *Commit* state. The L1 cache controller then starts writing back the cache lines belonging to the write set of the committed transaction via the cache write-back interface. After writing back all the modified cache lines to persistent memory, DHTM transitions to the *Commit Complete* state.

Abort: To abort an ACID transaction DHTM clears the read-/write-set tracking structures and invalidates the speculative state. DHTM logically clears the log entries by writing an abort log record and transitions to the *Abort* state. Abort requires an *Abort Complete* state when supporting overflows, which is described in the next section.

Recovery: At the time of failure, a durable transaction can be in any state. The recovery manager does not have to do anything for transactions in *Active*, *Abort* or *Abort Complete* state as none of the updates of the transactions would have been written back in-place in persistent memory. For transactions in *Commit* state, the recovery manager recovers the transaction by replaying its redo log. Finally, for transactions in *Commit Complete* state, the recovery manager does not have to do anything, since the write set would have already been written back to persistent memory.

III. Handling Overflow

DHTM handles write-set overflow by allowing for cache lines belonging to the write set to be replaced from the L1 cache to the LLC. However, in order to enable conflict detection the coherence state of the cache line in the LLC is kept unchanged [4]. This ensures that the LLC continues to show the cache line as being owned by the core executing the transaction. For versioning, we also need a mechanism to identify all the cache lines that have overflowed from the L1 cache to the LLC. To this end, DHTM maintains an *overflow list* along with the redo log in memory. When a dirty cache line overflows from the L1 cache to the LLC, DHTM writes the address of the overflowed cache line to the overflow list. On a commit, DHTM uses the overflow list to identify cache lines belonging to the write set that have overflowed, and writes them back to persistent memory before transitioning to the *Commit Complete* state. For aborts, DHTM reads the overflow list and invalidates the corresponding LLC cache lines and transitions to the *Abort Complete* state.

The recovery procedure remains the same with overflows, since the cache lines belonging to the write set that overflowed the L1 cache are already present in the redo log. It is worth noting that, as opposed to existing proposals for supporting overflow DHTM requires only minor changes to the coherence protocol and shared caches.

IV. Evaluation

We evaluated the performance benefits of DHTM using the GEM5 simulator for an 8-core machine with a multi-banked last level cache [3]. We compare DHTM with SO, a design that supports both atomic visibility and durability in software and ATOM [5] that implements atomic visibility in software and uses the state-of-the-art hardware logging mechanism for atomic durability.

For a set of micro-benchmarks, DHTM provides a 61% improvement in throughput over SO as it leverages hardware support for both atomic visibility and durability thereby enabling higher concurrency and faster durability. In comparison to ATOM, DHTM provides a 26% improvement in throughput. DHTM not only benefits from better concurrency (because of HTM), but also because DHTM uses hardware redo logging which is faster than undo logging used in ATOM. The undo log design of ATOM suffers from the overhead of persisting data in-place in the commit critical path. For TPC-C DHTM improves the throughput by 88% over the SO and by 21% over ATOM.

References

[1] S. Harizopoulos, D. J. Abadi, S. Madden, and M. Stonebraker, "OLTP Through the Looking Glass, and What We Found There," in *SIGMOD*, 2008.

[2] V. Leis, A. Kemper, and T. Neumann, "Scaling HTM-Supported Database Transactions to Many Cores," *IEEE TKDE*, vol. 28, no. 2, 2016.

[3] A. Joshi, V. Nagarajan, M. Cintra, and S. Viglas, "DHTM: Durable Hardware Transactional Memory," in *ISCA*, 2018, http://homepages.inf.ed.ac.uk/s1372211/pub/isca18.pdf.

[4] K. E. Moore, J. Bobba, M. J. Moravan, M. D. Hill, and D. A. Wood, "LogTM: log-based transactional memory," in *HPCA*, 2006.

[5] A. Joshi, V. Nagarajan, S. Viglas, and M. Cintra, "ATOM: Atomic Durability in Non-volatile Memory through Hardware Logging," in *HPCA*, 2017.