LogSI-HTM: Log Based Snapshot Isolation in Hardware Transactional Memory

Lois Orosa and Rodolfo Azevedo

Institute of Computing, University of Campinas (UNICAMP)
{lois.orosa,rodolfo}@ic.unicamp.br

Abstract

In this paper we propose an early idea about a new hardware transactional memory system that implements snapshot isolation (SI) using logs. With this scheme, we avoid specific costly hardware resources (multiversion memory) to keep the snapshots, and maintain the advantages of snapshot isolation (low abort rates). In our proposal, the aborting process is slower than the related works, but as the abort rate is enormously reduced with SI, the performance lost should be minimal.

1 Introduction

Transactional memory systems with snapshot isolation make a snapshot of the valid shared memory data before starting any transaction, and each transaction works with that data until commit time. The most attractive feature of this implementation is that read-write conflicts do not cause a transaction to abort, as the transactions are logically reordered by reading the appropriate version of the data to not break the atomicity and consistency.

Therefore, under the aforementioned system, the abort process has to be activated only when two transactions have write-write conflicts. In the STAMP benchmarks, from 75% to 90% of the transactional aborts are caused by read-write conflicts [1], which shows the potential of this approach.

In SI, transactions always read data from a snapshot of data valid at the beginning of the transaction. Future updates made by other committed transactions are not visible.

However, snapshot isolation is not serializable because of the write skew anomaly. A write skew occurs if there exists an invariant whose component variables span multiple concurrent transactions and the transactions have disjoint write sets. There are some proposals in the literature to solve this problem [1, 2], but it is not the focus of this paper.

In this work we propose an alternative snapshot isolation proposal in hardware transactional memory that presents a different approach with low overhead compared with the state of the art.

2 Related work: SI-TM

The first work proposing a hardware snapshot isolation was SI-TM [1]. For supporting memory snapshots, the authors introduce a new memory subsystem (implemented in the last level cache) that incorporates timestamps to store multiple versions of the same data. It supports the generation of new data versions on the flight, and introduces a hardware memory overhead from 12.5% to 50% (depending on the specific configuration and the implemented optimizations).

SI-TM implements lazy conflict detection, performing validation (look for write-write conflicts) at commit time. SI-TM further improves on existing lazy systems by introducing a new validation technique based on timestamps. This technique enables local commits, as transactions can validate their write set by comparing it against the state of main memory instead of broadcasting it to the other cores in the system.

In SI-TM, every transaction obtains unique start and end timestamps which are used by the Multiversion Memory (MVM) to locate the correct version. On a non-transactional read access, the MVM returns the newest version. Non-transactional writes modify the most current version in place. SI-TM provides architectural support for version management and allocation of shared data. Therefore, the main memory is partitioned into conventional memory and multiversion memory.

To address write skews, the authors of SI-TM also developed a best-effort technique based on dynamic code analysis which resulted in a tool that is able to handle large applications. The tool is implemented using PIN and instruments transactional memory applications at runtime.

3 Our proposal: LogSI-HTM

The baseline of LogSI-HTM has many elements in common with LogTM-SE [3]. LogTM-SE is an eager hardware transactional memory system that keeps the old versions of the data in a per-thread log, and tracks the conflicting addresses with Bloom Filters (leveraging the cache coherence protocol). At transaction commits, the log is discarded, and the Bloom filters are cleared. At transaction aborts, the old versions are copied to their place, and the log and Bloom filters are cleared.

LogSI-HTM takes from LogTM-SE the per-thread log for old versions of data, and conflict detection with Bloom filters (and leveraging the cache coherence protocol). Many of the hardware additions of LogSI-HTM are shared with LogTM-SE (including log pointer, log base, a Wtx bit per cache line, etc. See LogTM-SE[3] paper for more details).

For emulating the memory snapshots required in each transaction, LogSI-HTM keeps the old versions of data in a per-thread log until are not useful anymore by any transaction. Unlike LogTM-SE, our proposal associates a timestamp with each log, corresponding to the commit of the transaction. Furthermore, the all logs are shared among all the cores (read only) to access to the old version of data (according to their snapshots). Moreover, unlike LogTM-SE, the log is not cleared until there is not any in-flight transaction that could potentially use that version of the data.

The key point of our approach is that the system can emulate a snapshot of the memory by just accessing the old versions of data kept in the logs.

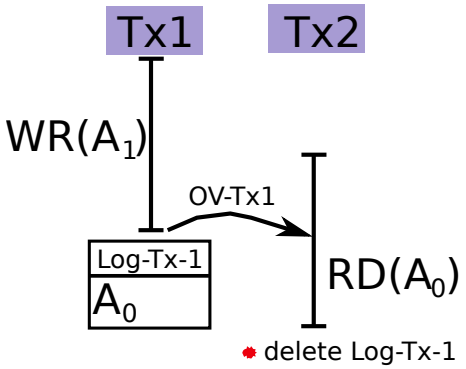
3.1 The Details

Each transaction has a timestamp associated with the beginning of the transaction. Furthermore, during the transaction, LogSI-HTM tracks the write-set with a Bloom filter , and unlike the original LogTM-SE, LogSI-HTM does not need to track transactional reads.

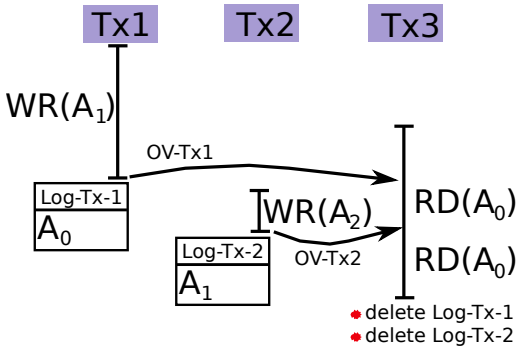
There are two new hardware elements in LogSI-HTM compared with the LogTM-SE. The first is a new **OV Bloom filter**, that keeps the addresses of the write sets of the transactions that commit while the current transaction was in flight. This Bloom filter is checked to test out if the transaction has to access the old data maintained in a log. We could reuse the read set Bloom filter of the original LogTM-SE to implement this OV Bloom filter. In case a transaction receives several write-sets from different committed transactions, they are merged into a unique OV Bloom filter. The second component is a hardware structure that maintains the **log pointers and timestamps** associated with all the write sets maintained in the OV Bloom filter.

To illustrate the behavior of LogSI-HTM, we will use the examples of Figure 1a and Figure 1c. Figure 1a shows two transactions (Tx1 and Tx2), the first writing the addr A, and the second reading also the addr A (the subindexes represents the different versions of the content of A along the time). Similarly, Figure 1c represents an example with 4 transactions, Tx1 and Tx2 writing in A, and Tx3 and Tx4 reading from A. In both figures Log-Tx-1 and Log-Tx-2 represent the content of the log at the end of the transaction. Also, OV-Tx1 and OV-Tx2 represent the OV Bloom Filters.

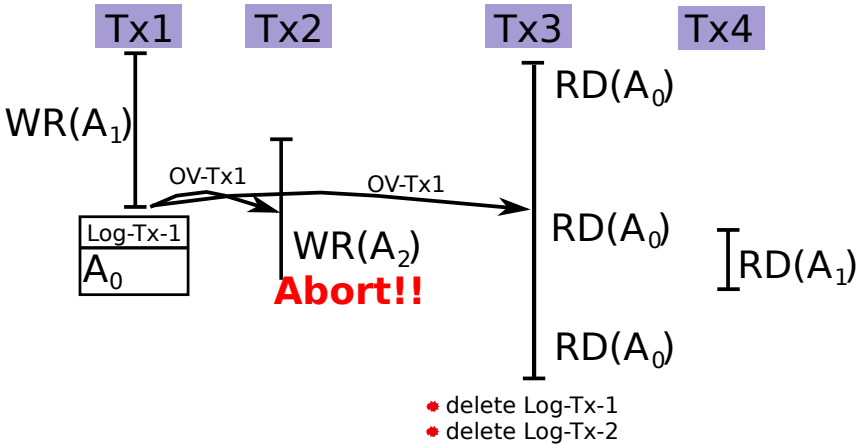
When a transaction commits, the log with the old versions of data is not discarded and it is associated with all the on-flight transactions at that moment. The last associated transaction to finish is in charge of clearing this log. This is an easy and effective way to do garbage collection of the old version no longer needed. In the example of Figure 1a, the log of transaction Tx1 is deleted



(a) Example with 2 transactions without aborts. Commit order: Tx1, Tx2 . Logical order: Tx2, Tx1.



(b) Example with 3 transactions without aborts. Commit order: Tx1, Tx2, Tx3 . Logical order: Tx3, Tx1, Tx2.



(c) Example with 4 transactions with one w-w conflict. Commit order: Tx1, Tx4, Tx3. Logical order: Tx3, Tx1, Tx4.

Figure 1: Examples for illustrating the algorithm.

by Tx2 when it commits. Furthermore, a committed transaction sends several informations to all the on-flight transactions: the write-set (content of the Bloom filter), a pointer to its log and the timestamp. The in-flight transactions save the write-set of the committed transaction in the OV Bloom filter, as well as the log pointer and the timestamp.

In each **transactional read**, the OV Bloom filter is checked. If there is no match in the OV Bloom filter, the read is performed as usually. If there is a match, it means that the transaction should read the value from one of the logs pointed by their log pointers (from already committed transactions). In Figure 1a, it is the case of Tx2 when reads address A: the address is in its OV Bloom filter, and therefore it has to read an old value from Log-Tx-1.

In case there are different versions of the same data in different logs, it just pick the one with the lower timestamp (to ensure that it is getting the right version of the data). The example of the Figure 1b illustrates why: at the time of the second read in Tx3, the transaction has the information of two logs, and it has to choose the older log to keep coherent with the first read made by the transaction. If the situation was different, and Tx3 only performed the second read, both values of A_0 would be correct.

Furthermore, if there is a running remote transaction that already wrote to the same address, the cache coherence protocol detects this situation (in the same way that LogTM-SE [3]), and instead of aborting (like in LogTM-SE), the remote core sends a response informing about the address of the old value saved in its log. This is necessary because the system implements eager version management, which implies that the speculative data is saved in place. In the traditional LogTM-SE this situation would cause an abort, but do not in snapshot isolation. In Figure 1c, it is the case of the read in transaction Tx4 after the write in Tx2. As this could be a potential

source of slowdown, we propose an optimization in Section 3.2.

In each **transactional write**, the OV Bloom filter is also checked. If there is a match (w-w conflict), the transaction has to abort to preserve the atomicity of the transactions. It is the case of Tx2 in Figure 1c.

Unlike SI-TM [1], our LogSI-HTM performs the conflict detection eagerly, because a write-write conflict almost always should cause an abort. Delaying the conflict detection until commit time (lazy conflict detection) would not result in any advantage for avoiding write-write conflicts (unlike read-write conflicts with a lazy conflict detection). However, there are some situations where we could avoid the abort in non critical conflicts, including false sharing, silent stores and write-write conflicts without intermittent reads [4].

3.2 Performance Considerations

As LogSI-HTM has so many elements in common with LogTM-SE, it also shares the same characteristics: the commits are fast (the speculative data are already in place at commit time), and the aborts are slow (the old data is in the log, and it has to be restored by software). However, as demonstrated in [1], the aborts are reduced enormously with snapshot isolation, as the most of the conflicts are read-write conflicts.

Other fonts of slowdown are when a read has to be done from a log instead of memory (the address is in the local OV Bloom filter, or the cache coherence protocol detects a read-write conflict). In this case, the system has the pointer to the log, but not the pointer to the specific data. Finding the specific data in the log efficiently will require efficient log organization and search algorithms.

To alleviate the read latency in this situation, we propose to use a **translation table** that correlates the data address with the old data of a specific version. This hardware table should be maintained locally (and updated in each transactional write), and sent to the in-flight transactions when commit. Usually the write set is not very big, but this table has to be of limited sized, and the data that is not in that table has to be software accessed (slower).

4 Conclusions and Future Work

In this paper we propose an implementation of snapshot isolation in hardware transactional memory with the next contributions:

- We propose an alternative implementation to the lazy SI-TM [1]: an eager conflict detection and eager version manager snapshot isolation transactional memory system.
- Our proposal does not require a special multiversion memory for keeping several versions of the data (simplification of the hardware).
- The garbage collection mechanism (to discard the old data when it is not needed anymore) is simple and effective.

As this is an early work, the future work is still extensive:

- Simulate our proposal in a cycle-accurate simulator.
- Evaluate the performance worries exposed in Section 3.2.
- In case they constitute a performance problem, propose new optimizations, protocols and/or hardware support (Section 3.2).
- Propose hardware alternatives to solve the problem of the write skew anomalies.

Acknowledgments

This work was supported by grants 2014/03840-2 and 2013/08293-7, São Paulo Research Foundation (FAPESP).

References

[1] H. Litz, D. Cheriton, A. Firoozshahian, O. Azizi, and J. P. Stevenson, "Si-tm: Reducing transactional memory abort rates through snapshot isolation," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '14. New York, NY, USA: ACM, 2014, pp. 383–398. [Online]. Available: http://doi.acm.org/10.1145/2541940.2541952

[2] R. J. Dias, J. M. Lourenço, and N. M. Preguiça, "Efficient and correct transactional memory programs combining snapshot isolation and static analysis," in *3rd USENIX Conference on Hot Topics in Parallelism (HotPar 2011). Usenix Association*, 2011.

[3] L. Yen, J. Bobba, M. R. Marty, K. E. Moore, H. Volos, M. D. Hill, M. M. Swift, and D. A. Wood, "Logtm-se: Decoupling hardware transactional memory from caches," in *Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture*, ser. HPCA '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 261–272. [Online]. Available: http://dx.doi.org/10.1109/HPCA.2007.346204

[4] M. Waliullah and P. Stenstrom, "Classification and elimination of conflicts in hardware transactional memory systems," in *Computer Architecture and High Performance Computing (SBAC-PAD), 2011 23rd International Symposium on*, Oct 2011, pp. 96–103.