WTTM 2012, The Fourth Workshop on the Theory of Transactional Memory

Vincent Gramoli
University of Sydney
Australia
vincent.gramoli@sydney.edu.au

Alessia Milani
Université de Bordeaux-IPB-Labri
France
milani@labri.fr

Abstract

In conjunction with PODC 2012, the TransForm project (Marie Curie Initial Training Network) and EuroTM (COST Action IC1001) supported the 4th edition of the Workshop on the Theory of Transactional Memory (WTTM 2012). The objective of WTTM was to discuss new theoretical challenges and recent achievements in the area of transactional computing. The workshop took place on July 19, 2012, in Madeira, Portugal.

This year's WTTM was a milestone event for two reasons. First, because the same year, the two seminal articles on hardware and software transactional memories [15, 21] were recognized as outstanding papers on principles of distributed computing, whose significance and impact on the theory and practice of distributed computing have been evident for at least a decade. Second, the winners of this prestigious ACM/EATCS Edsger W. Dijkstra Prize, Maurice Herlihy, Eliot Moss, Nir Shavit and Dan Touitou, were present at the workshop and three of them discussed their current progress with other outstanding researchers from the field. This report is intended to give highlights of the problems discussed during the workshop.

Transactional memory is a concurrency control mechanism for synchronizing concurrent accesses to shared memory by different threads. It has been proposed as an alternative to lock-based synchronization to simplify concurrent programming while exhibiting good performance. The sequential code is encapsulated in transactions, which are sequences of accesses to shared or local variables that should be executed atomically by a single thread. A transaction ends either by *committing*, in which case all of its updates take effect, or by *aborting*, in which case, all its updates are discarded and never become visible to other transactions.

1 Consistency criteria

Since the introduction of the transactional memory paradigm, several consistency criteria have being proposed to capture its correct behavior. Some consistency criteria have been inherited from

the database field (e.g., serializability, strict serializability), others have been proposed to extend these latter to take into account aborted transactions, e.g., opacity, virtual world consistency; some others have been proposed to define the correct behavior when transactions have to be synchronized with non transactional code, e.g., strong atomicity.

Among all the criteria, opacity, originally proposed by Guerraoui and Kapalka [13], gained a lot of attention. In his first talk, Srivatsan Ravi showed that in any safe restriction of opacity, no transaction may read from a transaction that has not invoked *tryCommit*. He presented a definition of opacity that intuitively captures this feature and showed that it is a safety property in the formal sense, i.e., it is prefix-closed and limit-closed. This is a joint work with Hagit Attiya, Sandeep Hans and Petr Kuznetsov.

Victor Luchangco presented his joint work with Mohsen Lesani and Mark Moir provocatively titled "Putting opacity in its place" in which he presented the TMS1 and TMS2 consistency conditions and clarified their relationship with the prefix-closed definition of opacity. Broadly, these conditions ensure that no transaction observes the partial effects of any other transaction in any execution of the STM without having to force a total-order on transactions that participate in the execution. In particular, TMS1 is defined for any object with a well-defined sequential specification while TMS2 is specific for read-write registers. They further show using IO Automata [18] that every behavior allowed by TMS2 is allowed by TMS1 and formally prove that the NOrec algorithm satisfies TMS2. Moreover, algorithms that satisfy opacity also satisfy TMS1, and algorithms that satisfy TMS2 also satisfy opacity.

While opacity defines the correctness of transactional memories when shared variables are accessed only inside transactions, understanding the interaction between transactions and locks or between accesses to shared variables in and outside transactions has been a major question. This is motivated by the fact that the code written to work in a transactional system may need to interact with legacy code where locks have been used for synchronization. It has been shown that replacing a lock with a transaction does not always ensure the same behavior.

Srivatsan Ravi presented his joint work with Vincent Gramoli and Petr Kuznetsov on the locally-serializable linearizability (ls-linearizability) consistency criterion that applies to both transaction-based and lock-based programs, thus allowing to compare the amount of concurrency of a concurrent program [10]. In short, this consistency criterion captures the correctness of high-level data types (linearizability) with the correctness of their low-level implementations (thread-safety) by guaranteeing that the sequence of sequential events corresponding to each high-level operation is consistent with "some" sequential execution.

Stephan Diestelhorst presented his preliminary work with Martin Pohlack, "Safely Accessing Timestamps in Transactions". He presented scenarios in which the access to the CPU timestamp counter inside transactions could lead to unexpected behaviors. In particular, this counter is not a transactional variable, so reading it inside a transaction can lead to a violation of the single lock atomicity semantics: multiple accesses to this counter within transactions may lead to a different result than multiple accesses within a critical section. In this talk, a solution to prevent several transactions from accessing the timestamp concurrently was also sketched.

Annette Bienusa presented the definition of snapshot trace to simplify the reasoning on the correctness of TMs that ensure snapshot isolation. This is a joint work with Peter Thiemann [5].

Finally, Faith Ellen stated that despite the fact that a rich set of consistency criteria have been proposed there is no agreement on the way the semantics of a transactional memory has to be defined: operationally [14], as a set of executions [13, 22], or using automata [7].

2 Data structures for transactional computing

Eliot Moss' talk intended to explore how the availability of transactions as a programming construct might impact the design of data types. He gave multiple examples on how to design data types having in mind that these types are going to be used in transactions.

In a similar direction, Maurice Herlihy considered data types that support high-level methods and their inverse. As an example a set of elements supports the method to add an element, $add(x)$, and the method to remove an element from the set $remove(x)$. For these kind of data types, he discussed the possibility to apply a technique called *transactional boosting* which provides a modular way to make highly concurrent thread-safe data structures transactional. He suggested to distinguish the transaction-level synchronization with the thread-level synchronization. In particular, to synchronize the access to a linearizable object, non-commutative method calls have to be executed serially (e.g., $add(x)$ and $remove(x)$). Two method calls are commutative if they can be applied in any order and the final state of the object does not change. For example, $add(x)$ and $remove(x)$ do not commute, while $add(x)$ and $add(y)$ commute. Since methods have inverses, recovery can be done at the granularity of methods. This technique exploits the object semantics to synchronize concurrent accesses to the object. This is expected to be more efficient than STM implementations where consistency is guaranteed by detecting read/write conflicts.

3 Performance

Improving the efficiency of TMs has been a key problem for the last few years. In fact, in order for transactional memory to be accepted as a candidate to replace locks, we need to show that it has performance comparable to these latter.

In her talk Faith Ellen summarized the theoretical results on the efficiency of TMs. She stated that efficiency has been considered through three axes: properties that state under which circumstances aborts have to be avoided (permissiveness [11], progressiveness [12], etc); progress/liveness conditions and parallelizability. This latter is formalized through different variants of the disjoint access parallelism (DAP) property [16, 8, 2]. She also summarized the known impossibility results on TMs, parametrized by the consistency criterion, conditions under which transactions are allowed to abort and the nature of disjoint-access parallelism allowed by the implementation. The talk also summarized the lower bounds on the complexity to implement a combination of semantics, progress and DAP properties.

Mykhailo Laremko discussed how to apply known techniques (e.g., combining) to boost the performance of existing STM systems that have a central point of synchronization. In particular, in his joint work with Panagiota Fatourou, Eleftherios Kosmas and Giorgos E. Papadakis, they augment the NOrec transactional memory by combining and replacing the single global lock with a set of locks. They provide preliminary simulation results to compare NOrec and its augmented versions, showing that these latter perform better.

Nuno Diegues with João Cachopo [6] study how to extend a transactional memory to support nested transactions efficiently. The difficulty is to take into account the constraints imposed by the baseline algorithm. To investigate different directions in the design space (lazy versus eager conflict detection, multiversion versus single version etc), they consider the following transactional memories: JVSTM[9], NesTM [3] and PNSTM[4] . The performance of these algorithms were compared considering workloads without nested transactions and workloads with nested transactions. Nuno

Diegues shows that PNSTM's throughput is not affected by parallel nesting, while it is the case for the throughput of JVSTM and NesTM. In particular, NesTM shows the greater degradation of performance w.r.t. the depth of the nesting.

4 Legacy code and hardware transactional memory

The keynote by Nir Shavit was about his joint work with Yehuda Afek and Alex Matveev on "Pessimistic transactional lock-elision (PLE)" [1]. PLE is a non-speculative technique for automatic replacement of read-write locks by STM code in which each transaction is executed only once and never aborts. The main idea is to organize concurrency to avoid the synchronization overhead. In particular, write transactions execute sequentially and maintain a public undo log such that read operations can collect a snapshot of the memory using it. At commit time, writing transactions check that no transaction is reading the values they have to update. Then the old values are discarded. Their STM implementation offers significant performance improvements over read-write locks while performing only marginally worse than optimistic STMs like TL2 on some workloads. His talk further touched upon how to integrate such a mechanism to compliment hardware support like Intel's HLE. The inherent cost of lock elision remains an open problem.

Maged Michael gave a talk on IBM BlueGene/Q HTM features and performance characteristics [23]. Similar to the Intel HLE, he pointed out that there are no guarantees given on transaction progress and described the implementation of an STM on top of the BG/Q HTM.

5 Distributed transactional memory

Distributed transactional memory is the implementation of the transactional memory paradigm in a networked environment where processes communicate by exchanging messages. Differently from transactional memory for multicore machines, the networked environment needs to take into account the non negligible communication delays. To support local accesses to shared objects, distributed transactional memories usually rely on replication. Pawel T. Wojciechowski presented his joint work with Jan Konczak. They consider the problem of recovering the state of the shared data after some node crashes. This requests to write data into stable storage. Their goal was to minimize writes to stable storage, which are slow, or to do them in parallel with the execution of transactions. He presented a crash-recovery model for distributed transactional memory which is based on deferred update replication relying on atomic broadcast. Their model takes into account the tradeoff between performance and fault-tolerance. See [24, 17] for more information.

Sebastiano Peluso claimed that efficient replication schemes for distributed transactional memory have to follow three design principles: partial replication, genuineness to ensure scalability and support for wait-free read-only transactions. According to genuineness the only nodes involved in the execution of a transaction are ones that maintain a replica of an object accessed by the transaction. He claimed that genuineness is a fundamental property for the scalability of distributed transactional memory. This is a joint work with Paolo Romano and Francesco Quaglia [19]. Additional information can be found in [20].

6 Conclusion

While transactional memory has become a practical technology integrated in the hardware of the IBM BlueGene/Q supercomputer and the upcoming Intel Haswell processors, the theory of transactional memory still misses good models of computation, good complexity measures, agreement on the right definitions, identification of fundamental and useful algorithmic questions, innovative algorithm designs and lower bounds on problems. Upcoming challenges will likely include the design of new transactional algorithms that exploit the low-overhead of low-level instructions on the one hand, and the concurrency of high-level data types on the other hand.
For further information the abstracts and slides of the talks can be found at http://sydney.edu.au/engineering/it/~gramoli/events/wttm4.

Acknowledgements We are grateful to the speakers, to the program committee members of WTTM 2012 for their help in reviewing this year's submissions and to Panagiota Fatourou for her help in the organization of the event. We would like to thank Srivatsan Ravi and Mykhailo Laremko for sharing their notes on the talks of the workshop.

References

[1] Y. Afek, A. Matveev, N. Shavit. Pessimistic Software Lock-Elision. In Proceedings of the 26th International Symposium on Distributed Computing, DISC 2012.

[2] H. Attiya, E. Hillel, and A. Milani. Inherent Limitations on Disjoint-Access Parallel Implementations of Transactional Memory. In Proceedings of the 21nd ACM symposium on Parallelism in algorithms and architectures, SPAA 2009, pages 69–78.

[3] W. Baek, N. Bronson, C. Kozyrakis, and K. Olukotun. Implementing and evaluating nested parallel transactions in software transactional memory. In Proceedings of the 22nd ACM symposium on Parallelism in algorithms and architectures, SPAA 2010, pages 253-262.

[4] J. Barreto, A. Dragojevic , P. Ferreira, R. Guerraoui, and M. Kapalka. Leveraging parallel nesting in transactional memory. In Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP 2010, pages 91-100.

[5] A. Bieniusa. Consistency, isolation, and irrevocability in software transactional memory, Phd Thesis, 2011, http://www.freidok.uni-freiburg.de/volltexte/8382/.

[6] N. Diegues and J. Cachopo. Exploring parallelism in transactional workloads. Technical Report RT/16/2012, INESC-ID Lisboa, June 2012.

[7] S. Doherty, L. Groves, V. Luchangco, and M. Moir. Towards Formally Specifying and Verifying Transactional Memory, REFINE 2009, pages 245–261.

[8] F. Ellen, P. Fatourou, E. Kosmas, A. Milani, and C. Travers. Universal Constructions that Ensure Disjoint-Access Parallelism and Wait-Freedom. In Proceedings of the 31st Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, PODC 2012, pages 115–124.

[9] S. M. Fernandes and J. Cachopo. Lock-free and scalable multi-version software transactional memory. In Proceedings of the 16th ACM symposium on Principles and practice of parallel programming, PPoPP 2011, pages 179-188.

[10] V. Gramoli, P. Kuznetsov, S. Ravi. Brief announcement: From sequential to concurrent: correctness and relative efficiency. In Proceedings of the 31st Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, PODC 2012, pages 241–242.

[11] R. Guerraoui, T.A. Henzinger, and M. Kapalka, Permissiveness in Transactional Memories. In Proceedings of the 22nd International Symposium on Distributed Computing, DISC 2008, pages 305–319.

[12] R. Guerraoui and M. Kapalka. The Semantics of Progress in Lock-Based Transactional Memory. In Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009, pages 404–415.

[13] R. Guerraoui and M. Kapalka. Principles of Transactional Memory, Morgan Claypool, 2010.

[14] T. Harris, J. Larus, and R. Rajwar. Transactional Memory, Morgan Claypool,2010.

[15] M. Herlihy, E. Moss. Transactional Memory: Architectural Support for Lock-Free Data Structures. In Proceedings of the 20th Annual International Symposium on Computer Architecture, ISCA 1993, pages 289–300.

[16] Israeli and Rappoport, Disjoint-access-parallel implementations of Strong Shared Memory Primitives. In Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing, PODC 1994, pages 151–160.

[17] J. Kończak, N. Santos, T. Żurkowski, P. T. Wojciechowski and A. Schiper. JPaxos: State Machine Replication Based on the Paxos Protocol. Technical report EPFL-REPORT-167765, Faculté Informatique et Communications, EPFL, july 2011.

[18] M. Lesani, V. Luchangco, M. Moir: A Framework for Formally Verifying Software Transactional Memory Algorithms. In Proceedings of the 23rd International Conference on Concurrency Theory, CONCUR 2012, pages 516–530.

[19] S. Peluso, P. Romano, F. Quaglia. SCORe: a Scalable One-Copy Serializable Partial Replication Protocol. In Proceedings of ACM/IFIP/USENIX 13th International Middleware Conference, Middleware 2012.

[20] S. Peluso, P. Ruivo, P. Romano, F. Quaglia, and L. Rodrigues. When Scalability Meets Consistency: Genuine Multiversion Update Serializable Partial Data Replication. In Proceedings of the 32nd International Conference on Distributed Computing Systems, ICDCS 2012, pages 455–465.

[21] N. Shavit, D. Touitou. Software Transactional Memory. In Proceedings of the 14th Annual ACM Symposium on Principles of Distributed Computing, PODC 1995, pages 204–213.

[22] M. Spear, L. Dalessandro, Marathe, and M. Scott, Ordering-Based Semantics for Software Transactional Memory. In Proceedings of the 12th International Conference on Principles of Distributed Systems, OPODIS 2008, pages 275–294.

[23] A. Wang, M. Gaudet, P. Wu, J. N. Amaral, M. Ohmacht, C. Barton, R. Silvera, M. M. Michael. Evaluation of blue Gene/Q hardware support for transactional memories. In Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques, PACT 2012, pages 127-136.

[24] P. T. Wojciechowski, T. Kobus, M. Kokociski. Model-Driven Comparison of State-Machine-based and Deferred-Update Replication Schemes. In Proceedings of the 31st IEEE International Symposium on Reliable Distributed Systems, SRDS 2012.