Needed: Foundations for Transactional Memory

Hagit Attiya
Dept. of Computer Science, Technion
hagit@cs.technion.ac.il

"It's all very well in practice, but it will never work in theory"
(French management saying)

Transactional memory is a leading paradigm for designing concurrent applications for tomorrow's multi-core architectures. It follows and draws much inspiration from earlier research on concurrent data structures and concurrency control. Quite remarkably, it has succeeded in breaking out of the research community, and is being seriously considered by the industry—both as part of software solutions and as the basis for novel hardware designs.

But this success comes at a price, as every new research paper is now being judged by its immediate relevance and applicability to current technology, while ignoring the long-term development of foundations for this important area.

So, while there is a large body of practical work, the theoretical principles are still lacking: There are no agreed-upon concepts, and even the terminology is muddled; there is great confusion between specifications, policies and implementations; even when they exist, correctness properties do not distinguish between safety, liveness and performance.

This lack of foundations hinders communication and interaction, both within the community of researchers investigating concurrent data structures and in its interactions with other communities, most notably those investigating programming languages and verification.

Previous foundations developed for similar architectural advances include *concurrency control* theory [11] and consistency models for *distributed shared memory* [9]. This experience indicates what should be incorporated in a theory for transactional memory—and in a broader perspective, concurrent data structures.

The first ingredient are *specifications*. These should include clear interfaces, as well as definitions of safety and liveness properties. There are several possibilities for picking these properties, differing along several dimensions, including, most importantly, the following issues:

(1) Is the transaction stated as a list of atomic actions (like in classical database transactions) or more semantically as capturing a high-level operation (as in distributed shared memory)? This choice determines whether the data set of a transaction is explicitly stated (in the former case) or not; in turn, this has implications on detecting dependencies and conflicts among transactions.

(2) Are the intermediate states of the transaction observable and hence they should be consistent (as in *view serializability*) or should only the final state be checked for consistency?

(3) Should the real-time order among transactions be respected (as in *linearizability* [8] and *strict serializability* [10]) or not?

(4) What liveness properties should be demanded, namely, when are transactions guaranteed to terminate?

Obviously, these aspects are not always orthogonal, and they interact with performance specifications and issues of transactions' abort. There is initial study of these concerns [7], but further elucidation is needed and the consequences should be understood.

My belief is that the difficult choice between possibilities will not yield a single agreed-upon specification, but several alternatives corresponding to interesting combinations of the above dimensions. (As is the state of affairs in specifications of *group communication* middleware [4].) Even a clear choice between a small number of specifications will make feasible the derivation of verification techniques. This will let us address one of the most important challenges of multi-core programming, namely, validating that applications are indeed correct.

Comparison between various specifications, so as to choose the most appropriate for a certain set-up, will be assisted by *complexity measures*. Such measures should allow evaluating the worst case, as was done for distributed shared memories [3], perhaps even the average case. Because transactional memory is optimized for the *common case*, it is also important to figure out what is the best case and evaluate its cost.

The biggest question seems to be *what* to measure exactly? Since most transactional memory designs allow individual transactions to starve, they must be evaluated according to the overall performance. One suggestion is to measure their *Makespan* (the total time to complete a set of transactions) compared to their execution by a clairvoyant scheduler [6, 2], but this is an overly pessimistic measure that does not distinguish between designs that behave differently in practice. A recent proposal suggests taking into account the data conflicts when evaluating performance, allowing interferences only among closely conflicting transactions [1].

Once there are accepted complexity measures, they should be used to appraise proposed implementations. What would be even more beneficial would be the derivation of impossibility results, since these have the foremost implications on what can be achieved. This includes lower bounds on the costs of obtaining various safety and liveness properties, and trade-offs among them.

Developing such complexity measures will allow evaluating designs beyond existing technology. Currently, the performance of proposed designs is mostly measured through benchmarks. These benchmarks are often micro-scale and therefore, non-predictive for large-scale deployment [5]; moreover, it is not obvious what kind of workloads should be used even in full-scale benchmarks.

Finally, good theory will help us learn from the past, namely, by using results of previous research on similar topics like distributed shared memory and database concurrency control. It

also allows us to leave the lessons for the future, because while concurrent access to shared data will always be an innate factor of computing, technology and computer architecture will eventually be transformed in way that will make transactional memory obsolete and necessitate new solutions. When this time comes, good principles will allow exploiting ideas from current research to address future needs.

References

[1] Hagit Attiya. Concurrency and the Principle of Data Locality. *IEEE Distributed Systems Online*, 8(9), September 2007.

[2] Hagit Attiya, Leah Epstein, Hadas Shachnai and Tami Tamir. Transactional Contention Management as a Non-Clairvoyant Scheduling Problem In *Proceedings of the 25th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 308–315, 2006.

[3] Hagit Attiya and Jennifer L. Welch. Sequential consistency versus linearizability. *ACM Transactions on Computer Systems*, 12(2):91–122, May 1994.

[4] Gregory V. Chockler, Idit Keidar and Roman Vitenberg. Group communication specifications: a comprehensive study. *ACM Computing Surveys*, 33(4):427–469, December 2001.

[5] Aleksandar Dragojevic, Rachid Guerraoui and Michal Kapalka. Dividing Transactional Memories by Zero. To appear in *3rd ACM SIGPLAN Workshop on Transactional Computing*, 2008.

[6] Rachid Guerraoui, Maurice Herlihy and Bastian Pochon. Toward a Theory of Transactional Contention Management. In *Proceedings of the 24th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 258–264, 2005.

[7] Rachid Guerraoui and Michal Kapalka. On the Correctness of Transactional Memory. In *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2008, to appear.

[8] Maurice Herlihy and Jeanette M. Wing. Linearizability: a correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems*, 12(3):463–492, June 1990.

[9] Bill Nitzberg and Virginia Lo. Distributed shared memory: a survey of issues and algorithms. *Computer*, 24(8)52–60, August 1991.

[10] Christos H. Papadimitriou. The serializability of concurrent database updates. *Journal of the ACM*, 26(4)631–653, October 1979.

[11] Gerhard Weikum and Gottfried Vossen, *Transactional Information Systems*, Morgan Kaufmann, 2001.