Understanding Transactional Memory (Extended Abstract)

João Lourenço

CITI – Universidade Nova de Lisboa
Lisboa, Portugal
joao.lourenco@di.fct.unl.pt

Transactional Memory [3] (TM) is a new paradigm for concurrency control that brings the concept of transactions, widely known from the Databases community, into the management of data located in main memory. TM delivers a powerful semantics for constraining concurrency and provides the means for the extensive use of the available parallel hardware. TM uses abstractions that promise to ease the development of scalable parallel applications by achieving performances close to fine-grained locking while maintaining the simplicity of coarse-grained locking.

In this talk we start with an introduction to TM and try to answer the following questions: What is Transactional Memory? Why do we need it? How can it be used?

We then discuss some of the benefits and shortcomings of using TM as a programming paradigm. We demonstrate that, while TM may contribute to the development of concurrent programs with fewer errors, its usage does not imply by itself the full correctness of the program. TM programs may still suffer from both low-level and high-level data races. Low-level data races, commonly called simply by data races, result from the inadequate use of a protection mechanism when accessing a shared memory location, such as forgetting to define a code block as atomic. We describe a process that allows detecting low-level data races in TM by resorting to a data race detector for lock-based programs.

A program that is free from low-level data races is guaranteed to not have corrupted data, but no assumptions can be made about data consistency. High-level data races result from the interleaving of atomic code blocks defined with the wrong scope, such that the program's data consistency constraints do not hold anymore. We illustrate how to use the tool described in [4], which combines static analysis techniques with heuristics to detect and identify high-level data races in TM programs.

We then address another relevant issue for concurrent programming which may also affect TM programs: functionally correct applications can still underperform on the available hardware. Low performance frequently results from bad design and/or coding decisions, that limit the exploitation of concurrency in the application and its mapping into a specific hardware. Performance tuning frequently relies into a two-step approach, first executing the program under control of a monitor and collecting the relevant data into a log (trace) file, and then using a *post-mortem* tool to analyze this trace file and making inferences about the program's behavior, possibly producing statistics and other relevant information. This approach is also valid for TM [1].

In the talk we describe JTraceView [2], a performance-tuning tool for TM programs, which includes four main components: a monitoring infrastructure for collecting dynamic traces of TM programs, with minimal impact in the behavior of the observed program; a trace file processor, which filters, sorts and synchronizes the events in the trace; a trace file analyzer,which explores different approaches to synthetize the information available in the trace file; and the visualizer of the synthetized information (GUI).

We conclude with some remarks on the work that still lies ahead, until we achieve a complete support of the TM paradigm in the software development cycle. We need both, developing new tools and adapting existing ones to fit this new paradigm.

References

1. João Lourenço, Ricardo Dias, João Luís, Miguel Rebelo, and Vasco Pessanha. 2009. Understanding the behavior of transactional memory applications. In Proceedings of the 7th Workshop on Parallel and Distributed Systems: Testing, Analysis, and Debugging (PADTAD '09). ACM, New York, NY, USA, Article 3, 9 pages.
2. JTraceView. http://www-asc.di.fct.unl.pt/~jml/Software/JTraceView-0.1.tar.bz2
3. Rachid Guerraoui, Michał Kapałka. (2010) Principles of Transactional Memory. Synthesis Lectures on Distributed Computing Theory 1:1, 1-193.
4. Bruno Teixeira, João Lourenço, Eitan Farchi, Ricardo Dias, and Diogo Sousa. 2010. Detection of Transactional Memory anomalies using static analysis. In Proceedings of the 8th Workshop on Parallel and Distributed Systems: Testing, Analysis, and Debugging (PADTAD '10). ACM, New York, NY, USA, 26-36.