Machine Learning-based Self-adjusting Concurrency in Software Transactional Memory Systems

Diego Rughetti, Pierangelo Di Sanzo, Bruno Ciciani, Francesco Quaglia
DIAG, Sapienza Università di Roma

***Abstract*—Software Transactional Memories (STM) are a promising paradigm for parallel programming on multicore platforms. One of the problems of STM systems is the performance degradation that can be experienced when applications run with a non-optimal concurrency level, namely number of concurrent threads. When this level is too high a loss of performance may occur due to excessive data contention and consequent transaction aborts. Conversely, if concurrency is too low, the performance may be penalized due to limitation of both parallelism and exploitation of available resources. In this paper we will present an introduction to a complete work [1], which will be published in the proceedings of the IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer And Telecommunication Systems (MASCOTS 2012), in which we propose a machine-learning based approach which enables STM systems to predict their performance as a function of the number of concurrent threads in order to dynamically select the optimal concurrency level during the whole lifetime of the application. In our approach, the STM is coupled with a neural network and an on-line control algorithm that activates or deactivates application threads in order to maximize performance via the selection of the most adequate concurrency level, as a function of the current data access profile. A real implementation of our proposal within the TinySTM open source package and an experimental study relying on the STAMP benchmark suite has been realized. The experimental data confirm how our self-adjusting concurrency scheme constantly provides optimal performance, thus avoiding performance loss phases caused by non-suited selection of the amount of concurrent threads and associated with the above depicted phenomena.**

I. Introduction

Over the last decade multi-core systems have become mainstream computing architectures. Systems with up to 16 or 32 CPU-cores can be purchased for a few thousands dollars. This trend has lead to a growing need for the development of applications which can effectively exploit parallelism, thus bringing parallel programming out from the niche of scientific and high-performance computing. Within this context, Software Transactional Memories (STMs) [2] have emerged as a programming paradigm tailored for the development of concurrent applications. By leveraging on the concept of atomic transactions, historically used in field of database systems, STMs relieve programmers from the burden of explicitly writing complex, error-prone thread synchronization code. STMs provide a simple and intuitive programming model, where programmers wrap critical-section code within transactions, thus removing the need for using fine-grained lock-based synchronization approaches. Programmers' productivity is therefore improved, while not sacrificing the advantages provided by high parallelism. Data conflicts are handled within STMs by means of conflict detection and management algorithms, and most of the literature work made in this field has been aimed at designing increasingly effective conflict detection and management schemes [3], [4], [5], [6], [7]. None of the above approaches has been targeted at directly controlling and optimizing the level of parallelism, which would lead to the identification of suited values for the total amount of threads sustaining the application as a function of the workload profile. Increasing the number of concurrent threads can speed-up the application as more transactions (and more non-transactional code blocks) can be processed in parallel. But increasing the number of concurrent transactions typically causes an increase of the transaction conflict rate. As a consequence, transactions may experience more abort/retry phases, which give rise to an increase of the execution time. In general, it is not convenient to have more active threads than the available CPU-cores [8]. This is not sufficient to avoid the loss of performance due to excessive data contention. The choice of the well suited degree of concurrency is fundamental in order to obtain adequate trade-offs between parallelism and data conflict. Also, it is an orthogonal problem with respect to the data contention management. By exclusively considering the transaction rollback probability, several relevant parameters having a real impact on the actual transaction execution latency are not included, such as the workload profile of the running application. We want present an approach relying on machine learning, which also tackles the aforementioned shortcomings, where we use a neural network [9] to enable the prediction of the performance of STM applications as a function of the concurrency level (by also indirectly capturing the workload profile and hardware effects). The neural network is trained using a data set obtained by profiling the workload generated by the application. Then, at run time, a statistical characterization of the application workload is periodically generated, which is used by a control algorithm as input to the neural network in order to predict the wasted transaction execution times. These predictions, are finally exploited by the control algorithm to regulate the concurrency level with the aim at maximizing the application throughput. To evaluate the effectiveness of our self-adapting concurrency proposal, we have implemented the whole architecture by leveraging on TinySTM[4] and we have performed an extended experimental study by relying on the STAMP benchmark suite[10]. By the experimental data the overhead introduced by the implemented self-adaptive concurrency functionalities (vs the baseline case) reveals almost negligible, which, together with the effectiveness of the proposed machine-learning based prediction method, allow our solution to provide optimal performance across the whole set of tested workloads.

The work introduced in this paper is available at: http://www.dis.uniroma1.it/∼rughetti/publications/mascots12.pdf

II. Related Work

In [11] an analytical modeling approach has been proposed to evaluate the performance of STM applications as a function

of the number of concurrent threads and other workload configuration parameters. In this kind of approach a detailed knowledge of the specific conflict detection and management scheme used by the target STM is required, which is instead not required by the approach we are currently proposing. The work in [12] presents an analytical model taking as input a workload characterization of the application expressed in terms of transaction profiles, contention probability and hardware resources consumption. The model prediction is a representation of the average system behavior over the whole lifetime of the application. Hence, differently from our proposal, no ability to capture run-time variations is envisaged. The proposal in [13] is targeted at evaluating scalability aspects of STM systems. It relies on the usage of different types of functions to approximate the performance of the application when considering different amounts of concurrent threads. The approximation process is based on measuring the speed-up of the application over a set of runs, each one executed with a different number of concurrent threads, and then on calculating the proper function parameters by interpolating the measurements. Differently from our proposal, in this approach the workload profile of the application is not taken into account, hence the prediction may prove unreliable when the profile gets changed wrt the one used during interpolation phases. In [14] a control algorithm dynamically changes the number of threads which can concurrently execute transactions on basis of the observed transaction conflict rate. In the approach proposed in [15], incoming transactions are enqueued and sequentialized when an indicator, referred to as *contention intensity*, exceeds a pre-established threshold. In the proposal presented in [16], a transaction is sequentialized when a potential conflict with other running transactions is predicted. The prediction leverages on the estimation of the expected transaction read-set and write-set. Compared to our approach, all the above proposals do not directly estimate the wasted time due to aborted transactions. They only indirectly attempt to control the wasted time according to heuristics schemes. In [17] machine learning techniques are used to select the best performing conflict detection and management algorithm. In [18], machine learning is used to select the most suitable CPU-thread mapping. The goals of both these works are orthogonal with respect to our one, which focus on the regulation of the overall concurrency level in the system.

III. Conclusions and Future Work

In this paper we presented an introduction to a complete work, which will be published in the proceedings of the IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer And Telecommunication Systems (MASCOTS 2012), that proposes a novel machine learning-based solution addressing the problem of the dynamic selection of the optimal concurrency level in the context of STM systems. We experimented our approach by implementing the system architecture we introduced, thus building an STM which can self-adjust the concurrency level by activating and deactivating concurrent threads on the basis of the profile of the current workload. In our evaluation experiments we used applications selected from the STAMP benchmark suite. The results we got are very promising, as they shown that, in most of the cases, the performance achieved is, independently of the maximum number of concurrent threads of the application, close to the best case when using a fixed (optimal) number of running threads. In particular, we observed that when an application is executed with an overestimated number of concurrent threads, our self-adjusting STM proves to be able to reduce the concurrency level so to avoid the typical performance degradation experienced with traditional (non-self adjusting) STM systems.

References

[1] Diego Rughetti, Pierangelo Di Sanzo, Bruno Ciciani, and Francesco Quaglia. Machine learning-based self-adjusting concurrency in software transactional memory systems. In *Proceedings of the 20th IEEE International Symposium On Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 2012.

[2] Nir Shavit and Dan Touitou. Software transactional memory. In *Proceedings of the fourteenth annual ACM symposium on Principles of distributed computing*, pages 204–213, New York, NY, USA, 1995. ACM.

[3] Dave Dice, Ori Shalev, and Nir Shavit. Transactional Locking II. In *In Proc. of the 20th Intl. Symp. on Distributed Computing*, 2006.

[4] Pascal Felber, Christof Fetzer, and Torvald Riegel. Dynamic performance tuning of word-based software transactional memory. In *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*, pages 237–246, New York, NY, USA, 2008. ACM.

[5] Maurice Herlihy and J. Eliot B. Moss. Transactional memory: architectural support for lock-free data structures. *SIGARCH Comput. Archit. News*, 21(2):289–300, May 1993.

[6] Michael F. Spear, Luke Dalessandro, Virendra J. Marathe, and Michael L. Scott. A comprehensive strategy for contention management in software transactional memory. In *Proceedings of the 14th ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 141–150, New York, NY, USA, 2009. ACM.

[7] Yossi Lev, Victor Luchangco, Virendra J. Marathe, Mark Moir, Dan Nussbaum, and Marek Olszewski. Anatomy of a scalable software transactional memory. In *2009, 4th ACM SIGPLAN Workshop on Transactional Computing (TRANSACT09*, 2009.

[8] Robert Ennals. Software transactional memory should not be obstruction-free. Technical Report IRC-TR-06-052, Intel Research Cambridge Tech Report, Jan 2006.

[9] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[10] Chi C. Minh, Jaewoong Chung, C. Kozyrakis, and K. Olukotun. STAMP: Stanford Transactional Applications for Multi-Processing. In *Proceedings of the IEEE International Symposium on Workload Characterization*, pages 35–46, Seattle, WA, USA, 2008.

[11] Pierangelo Di Sanzo, Bruno Ciciani, Roberto Palmieri, Francesco Quaglia, and Paolo Romano. On the analytical modeling of concurrency control algorithms for software transactional memories: The case of commit-time-locking. *Performance Evaluation*, 69(5):187 – 205, 2012.

[12] Zhengyu He and Bo Hong. Modeling the run-time behavior of transactional memory. In *Modeling, Analysis Simulation of Computer and Telecommunication Systems, 2010 IEEE International Symposium on*, pages 307 –315, aug. 2010.

[13] Aleksandar Dragojević and Rachid Guerraoui. Predicting the Scalability of an STM: A Pragmatic Approach, 2010.

[14] Mohammad Ansari, Christos Kotselidis, Kim Jarvis, Mikel Luján, Chris Kirkham, and Ian Watson. Advanced concurrency control for transactional memory using transaction commit rate. In *Proceedings of the 14th international Euro-Par conference on Parallel Processing*, pages 719–728, Berlin, Heidelberg, 2008. Springer-Verlag.

[15] Richard M. Yoo and Hsien-Hsin S. Lee. Adaptive transaction scheduling for transactional memory systems. In *Proceedings of the twentieth annual symposium on Parallelism in algorithms and architectures*, pages 169–178, New York, NY, USA, 2008. ACM.

[16] Aleksandar Dragojević, Rachid Guerraoui, Anmol V. Singh, and Vasu Singh. Preventing versus curing: avoiding conflicts in transactional memories. In *Proceedings of the 28th ACM symposium on Principles of distributed computing*, pages 7–16, New York, NY, USA, 2009. ACM.

[17] Qingping Wang, Sameer Kulkarni, John V. Cavazos, and Michael Spear. Towards applying machine learning to adaptive transactional memory. In *Proceedings of the 6th ACM SIGPLAN Workshop on Transactional Computing*, 2011.

[18] C.P Ribeiro M. Cole M. Cintra M. Castro, L.F. Wanderley-Goes and J. Mehaut. A machine learning-based approach for thread mapping on transactional memory applications. In *Proceedings of the 18th Annual International Conference on High Performance Computing*, 2011.