2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing

# Evaluating the Impact of Transactional Characteristics on the Performance of Transactional Memory Applications

Fernando Rui*, Márcio Castro†, Dalvan Griebler*, Luiz Gustavo Fernandes*
*GMAP Research Group (FACIN/PPGCC) – PUCRS, Brazil
Email: fernando.rui, dalvan.griebler{@acad.pucrs.br}, luiz.fernandes@pucrs.br
†Institute of Informatics – Federal University of Rio Grande do Sul (UFRGS), Brazil
Email: mbcastro@inf.ufrgs.br

***Abstract*—Transactional Memory (TM) is reputed by many researchers to be a promising solution to ease parallel programming on multicore processors. This model provides the scalability of fine-grained locking while avoiding common issues of traditional mechanisms, such as deadlocks. During these almost twenty years of research, several TM systems and benchmarks have been proposed. However, TM is not yet widely adopted by the scientific community to develop parallel applications due to unanswered questions in the literature, such as "how to identify if a parallel application can exploit TM to achieve better performance?" or "what are the reasons of poor performances of some TM applications?". In this work, we contribute to answer those questions through a comparative evaluation of a set of TM applications on four different state-of-the-art TM systems. Moreover, we identify some of the most important TM characteristics that impact directly the performance of TM applications. Our results can be useful to identify opportunities for optimizations.**

***Keywords*-Transactional memory; performance evaluation;**

## I. Introduction

The multicore technology has proven to be able to accomplish high performance along with reduced power consumption. However, applications must be parallelized in such a way that the degree of concurrency is optimized to exploit the full power of multicores. Unfortunately, most of the problems in high performance computing are not embarrassingly parallel, so developers often rely on synchronization mechanisms to guarantee the correct execution of concurrent accesses on shared data. Traditional synchronization structures such as locks, mutexes and semaphores are extensively used in a multicore context. However, they have several disadvantages: (i) they are low-level mechanisms, since one must explicitly control the access to shared variables; (ii) they cause blocking, so threads always have to wait until a lock (or a set of locks) is released; (iii) they are hard to manage effectively, especially in large systems; and (iv) they can be vulnerable to failures and faults, such as deadlocks and livelocks [1].

In this context, Transactional Memory (TM) provides a new attractive way of developing parallel applications through a higher abstraction level, shifting the synchronization problem to the TM system, which is responsible for ensuring that deadlocks will not occur and race conditions are correctly handled [2], [3]. It allows programmers to write parallel code as transactions, which are guaranteed to execute atomically and in isolation regardless of eventual data races [1]. At runtime, transactions are executed speculatively and the TM runtime system continuously keeps track of concurrent accesses and detects conflicts. Conflicts are then solved by re-executing conflicting transactions. TM can be implemented in hardware (HTM), software (STM) or both (HyTM). Our work focuses on STM systems because they are easier to modify and have no architectural limitations compared to hardware [4].

Several studies were carried out to improve the use of TM in parallel programming during the last twenty years. In addition to that, different TM systems [5], [6], [7], [8] as well as TM benchmarks from different domains [9], [10] were proposed. Although there have been advances in the area, TM is still under research due to unanswered questions. As discussed in [11], it is still challenging to know what kind of applications can really take advantage of TM. Finally, as identified by [12], it is essential to investigate the reasons why some TM applications present low performance and how to identify in advance which applications can benefit from this model.

Considering this scenario, this work presents a comparative evaluation of a set of transactional applications and systems. We intend to take a step towards understanding the existing problems and identifying opportunities in the STM systems in order to contribute to answer the questions that remain open. Our main goals are to present a comparative analysis of STM systems and to identify the characteristics of TM applications that impact the most on the performance of STM systems. Our contributions can be summarized as follows:

1) We present a performance evaluation of state-of-art STM systems and TM applications;
2) We extend the analysis presented in [10], including the RSTM [7] system;
3) We find out some transactional characteristics that considerably affect the performance TM applications;
4) We identify some of the bottlenecks of TM applications that limit their scalability and we show possible improvements to achieve better performance.

The rest of this paper is organized as follows. In Section II, we introduce our research methodology. A performance analysis of STM systems are presented in Section III and IV. In Section V, we analyze the impact of transactional characteristics on the performance of TM applications. Section VI reviews some related works concerning the performance evaluation of STM systems. Finally, concluding remarks and future works are presented in Section VII.

## II. Methodology

Our main goal is to present a comparative analysis of STM systems and to identify the characteristics of TM applications that impact the most on their performances. We caried out three sets of experiments to achieve this goal.

First, we analyze the performance of four state-of-the-art STM systems using the Stanford Transactional Applications for Multi-Processing (STAMP) benchmark [9]. Second, we perform a thorough evaluation of STM systems using EigenBench [10], which can mimic the behavior of STAMP applications while offering a fine-tune control of the input parameters. As shown in [10], EigenBench can attain very similar transactional behavior of STAMP applications. Finally, we evaluate the impact of certain transactional characteristics on the performance of TM applications using EigenBench. This experiment allows us to study the behavior of applications in a controlled manner by changing one characteristic at a time.

All experiments were performed on a Dell PowerEdge R610 machine with two quad-core Intel Xeon E5520 2.27 GHz processors with 8MB of L2 cache and 16GB of shared memory. All results are arithmetic means of at least 30 runs to guarantee a confidence level of 95%.

## III. Performance Comparison of STM Systems Using STAMP Benchmark

In this work, we focus on STM systems implemented in C/C++. We first give a brief description of the STM systems used in this study. Then, we evaluate the performance obtained with these STM systems on STAMP applications.

### A. STM Systems

STM systems such as TL2 [8], TinySTM [5], SwissTM [6] and RSTM [7] are examples of state-of-the-art STM systems. In this paper, we used the latest versions available of these systems and compiled them with their standard configurations.

The Transactional Locking II (TL2) is the second version of the original Transactional Locking (TL) algorithm developed by D. Dice and N. Shavit [8]. TinySTM [5] is another well-known STM implementation that also uses a global versioning approach (shared counter as clock) to control the conflicts between transactions and locks to protect shared memory locations. SwissTM [6] presents some new features when compared to TL2 and TinySTM. One of its innovations is the hybrid conflict detection scheme: it detects write/write conflicts eagerly, which prevents transactions that will probably abort from running and wasting resources, and read/write conflicts lazily, allowing more parallelism between transactions. In read/write conflicts, a time-based scheme (similar to the TL2 global version-clock) is applied to handle conflicts. Finally, the Rochester Software Transactional Memory (RSTM) [7] is one of the oldest open-source STM systems. RSTM reduces cache misses by employing a single level of indirection to access shared objects. It means that each object has a unique metadata structure during its lifetime, avoiding the creation of a new locator whenever a object is acquired by a transaction.

### B. Performance Evaluation

The STAMP benchmark suite includes 8 applications and 30 variants of input parameters and data sets. We used the same set of input parameters for each application for non-simulated runs as presented in [9]. Figure 1 presents speedups of each STAMP application. Speedups are relative to a sequential baseline without transactions.

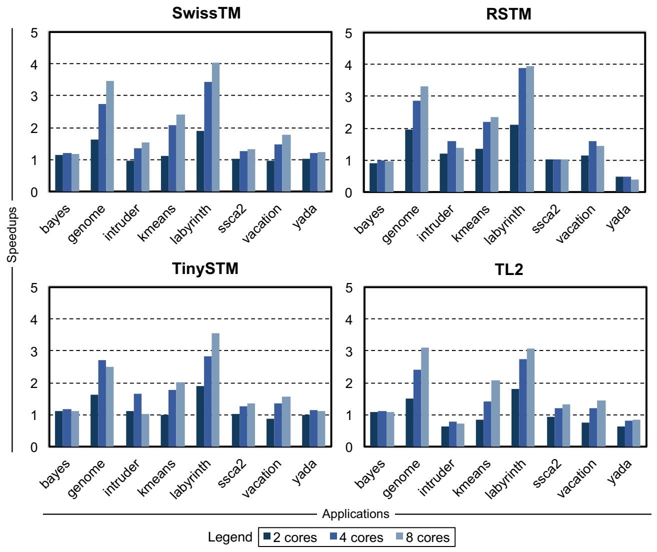

Figure 1. Speedups of STAMP applications.

Overall, SwissTM and RSTM achieved better performance than other STM systems. However, all systems presented poor results when running *ssca2*, *yada* and *bayes*. After analyzing the source codes, of these applications we identified several regions of non-transactional code. Although this limits the parallelism, it is necessary to investigate the transactional characteristics of such applications to identify reasons for their poor performance.

Among all applications, only two achieved good scalability. *Labyrinth* presented ideal performance with 2 cores on all STM systems, but with 4 cores only RSTM and SwissTM sustained a good performance. Although, *genome* achieved good performance with 2 cores on all systems, only RSTM achieved ideal speedup. For more than 4 cores, RSTM and SwissTM systems showed better overall performances.

Even though the STAMP benchmark has a set of applications that stresses out STM systems, we could verify that all tested STM systems could not achieve satisfactory results in terms of performance. In fact, the characteristics of the STAMP applications presented in [9] do not give enough information to fully comprehend their performance results. A good starting point to better understand these performance issues would be to find a way to identify and manipulate the transactional characteristics of STAMP applications. These characteristics must represent the application as completely as possible, indicating attributes that may help to understand their performance and to identify opportunities for improvements. We further study these aspects in Section V.

## IV. SwissTM *vs.* RSTM using EigenBench

In this experiment, we narrowed our set of tested STM systems to those which presented better performance in the previous experiment in Section III, *i.e.,* SwissTM and RSTM. The reasons for choosing two STM systems instead of only the best one are threefold: (i) these systems showed very similar results for some STAMP applications; (ii) the use of two systems will allow us to compare their results helping to detect unusual behaviors; and (iii) RSTM is an important system in the literature and was not yet tested with EigenBench [10]. We also decided to reduce the number of STAMP applications for this experiment. We chose applications with poor (*ssca2*), medium (*intruder* and *vacation*) and good (*labyrinth* and *genome*) scalability.

Our evaluation is based on two relevant metrics used in the context of TM [13]: speedup and aborts per commit (ApC). The former focus on the scalability of the application, which will be in fact the overall performance of the TM application. The latter, on the other hand, reveals the degree of conflicts of the TM application and shows how well the STM system deals with those conflicts.

### *A. EigenBench Input Parameters*

In EigenBench, we can describe an application based on its Eigen characteristics. *Concurrency* is number of concurrently running threads; *Working-set Size* is size of frequently used memory; *Transaction Length* is number of shared accesses per transaction; *Pollution* is fraction of shared writes to shared accesses; *Temporal Locality* is probability of repeated address per shared access; *Contention* is probability of conflict of a transaction; *Predominance* is fraction of shared access cycles to total execution cycles; and *Density* is fraction of non-shared cycles executed outside transactions to total non-shared cycles.

The above mentioned Eigen characteristics are derived from a set of *input parameters* [10]: `A1, A2*N, A3, R1, W1, R2, W2, R3o, W3o, R3i, W3i` and `LCT`. Thus, to change the Eigen characteristics of a workload one needs to set all these input parameters.

Several different TM workloads can be created by carefully setting the input parameters described above. In fact, the authors of EigenBench proposed input parameters to mimic the behavior of STAMP applications [10]. We then use the same set of values for the input parameters and characteristics defined in [10] to create workloads very similar to those available on STAMP. Table I presents the input parameters used in our study. The *working-set size* and *transactional length* are average values.

Table I
Applications characteristics from STAMP benchmark

| Characteristic | ssca2 | intruder | vacation | labyrinth | genome |
|---|---|---|---|---|---|
| **Working-set Size** | 400 MB | 20 MB | 256 MB | 16 MB | 20 MB |
| **Transactional Lenght** | 3 | 24 | 226 | 357 | 88 |
| **Pollution** | 33% | 5% | 2% | 50% | 5% |
| **Temporal Locality** | 0.33 | 0.52 | 0.59 | 0.77 | 0.58 |
| **Contention** | 0.0005% | 22% | 0.2% | 5% | 0.5% |
| **Predominance** | Low | Low | High | Low | High |
| **Density** | High | High | High | Low | High |

### *B. Performance Evaluation*

Figure 2 shows the results using EigenBench to mimic the selected STAMP applications. Results with SwissTM are similar to those introduced in [10], confirming that we could reproduce their experiment correctly. As it can be observed, SwissTM outperforms RSTM in 4 out of 5 applications. Overall, applications presented a very similar behavior in terms of speedups with the single exception of *genome* with 8 cores, which was significantly different on both systems.

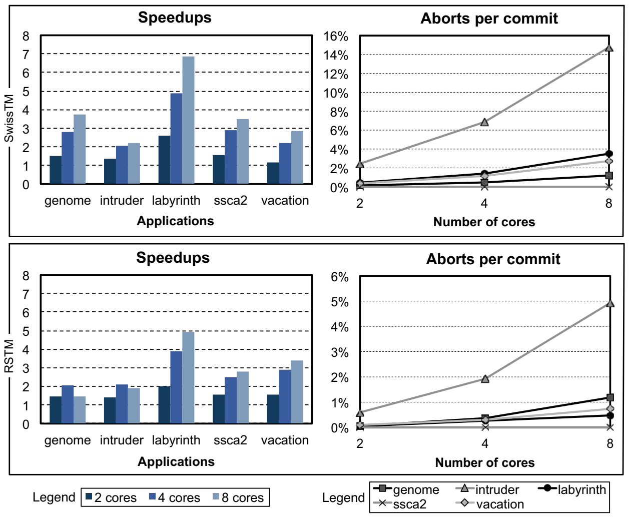

Figure 2. Speedups and ApCs of EigenBench on SwissTM and RSTM.

To perform the analysis of each individual application, we used the following information: speedups, ApC, orthogonal characteristics of each application and source codes. We discuss below our findings and conclusions.

**Genome.** It was the only application that presented significantly distinct performance results among STM systems. This fact is explained by the design choices of SwissTM which allows the system to deal satisfactorily with both short and long transactions. The application has different transaction lengths, ranging from very short transactions up to long transactions. Even though the fact that smaller transactions are more frequent than long ones in this benchmark, the use of longer transactions resulted in a reduction of the overall performance of the application. High predominance and high density also influenced its performance.

**Intruder.** It presented the worst speedups among the tested applications. This application has a large variation in terms of memory allocation during its execution time. The main reason for the poor performance is its high level of contention, which increased the number of ApCs (Figure 2). This contention is caused by the fact that the main shared structure (a self-balancing tree) has much less nodes near the end of the execution. Because of that, the probability of having transactions accessing the same nodes is very high, increasing considerably the ApC metric.

**Labyrinth.** It achieved the best results among the selected applications. The application has a uniform distribution in transaction length and the amount of memory used is often small. Although this application has large transactions, they have low density and low predominance. Additionally, this application presents low contention and high locality, which also contributed to its good performance.

**Ssca2.** It showed poor performance on both systems. Although the application has short transactions, it uses a large amount of memory which impacts the overall performance. Unlike *intruder*, it does not present high percentage of ApCs. This indicates that the loss of performance is not caused by conflicts. Instead, it comes from the difficulty of STM systems to deal simultaneously with both short transactions and large amounts of memory.

**Vacation.** It also presented poor performance on both systems. It has the following combination of characteristics that compromise its performance: a large amount of memory and a large variety of transaction lengths. Moreover, high predominance and high density are also present.

Our findings can be summarized as follows: (i) TM applications that use large amounts of memory did not present good performance, since STM systems need to keep track of much more data to detect conflicts; (ii) the variation in terms of transaction lengths during the execution is not well treated by most of the STM systems; (iii) low degrees of predominance and density help TM applications to perform better; and (iv) high levels of ApC generally limit the performance of TM applications.

## V. Evaluating the Impact of Transactional Characteristics

To perform this test, we selected the four applications that presented low performance in our experiments in Section IV: *genome*, *intruder*, *ssca2* and *vacation*. Our goal in this last experiment is to identify opportunities to improve the performance of these applications. We use a single STM system (*i.e.,* SwissTM), since we want to focus on the characteristics of TM applications. We performed a "trial and error" approach [10] driven by the knowledge acquired in our previous analysis. Based on that, we identified which transactional characteristics were relevant for each application. We then isolated them to analyze the impact of such characteristics on the performance of the TM applications. We present below the initial analysis of each application as well as the proposed modifications to improve their performances.

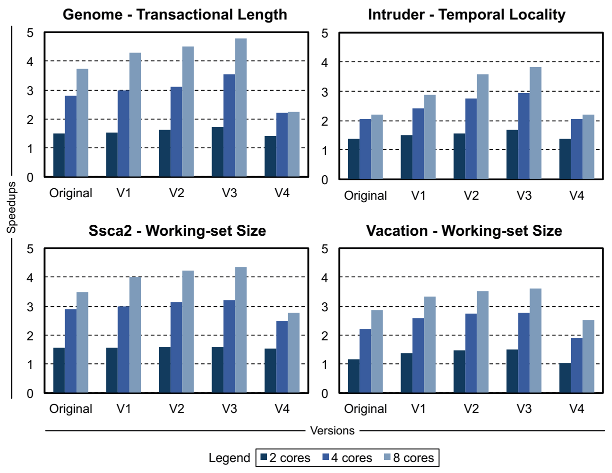

Figure 3. Speedups when varying transactional characteristics.

Among the characteristics discussed in Section IV, some results were relevant to determine the performance of applications. As an example, we could cite: *contention*, which impacted the result of *intruder*; *temporal locality*, which helped the performance of the *labyrinth*; *working-set size*, which impacted in applications with large memory (*ssca2* and *vacation*); and, *transactional length*, which impacted applications with both short and long transactions (*genome*). In the following sections, we analyze each one of the selected applications separately.

**Genome.** The poor performance of *genome* was strongly related to the *transaction length*. This characteristic is derived from following EigenBench input parameters: `R1`, `R2`, `W1` and `W2`. We varied these input parameters to analyze their performance impacts. Version V1 reduces the size of long transactions. Versions V2 and V3 reduce the sizes of both medium and long transactions. On the contrary, version V4 increases the sizes of both medium and long transactions. Figure 3-*genome* presents the results when varying these parameters. Versions V1 to V3 showed better results than the original application since we identified that long transactions lengths were the cause of poor performance in this application. Version V4 confirms our findings, since the performance of *genome* becomes worse than the original one when we increase the length of transactions.

**Intruder.** After carrying out several experiments, we found that the poor performance of *intruder* was due to *temporal locality* issues. This characteristic is directly related to the input parameter `LCT` in EigenBench. `LCT` can assume values ranging from 0 to 1,024. The higher is the value, the higher will be the probability of accessing a previous used memory address in a transaction. We generated four modified versions of *intruder* with different values for `LCT`. Versions V1, V2 and V3 increase the *temporal locality* whereas version V4 reduces it to the worst case possible.

In Figure 3-intruder, we confirm that the *temporal locality* has an important role on the performance of *intruder*, since versions V1, V2 and V3 presented better speedups.

**Ssca2 and vacation.** In Section IV-B, we concluded that the poor performance of *ssca2* and *vacation* came from the use of a large amount of memory inside transactions. EigenBench allows us to control it through the Eigen characteristic called *working-set size*. This characteristic is derived from the sum of the following input parameters: `A1`, `A2` and `A3`. Versions V1, V2 and V3 reduce the amount of memory used inside transactions whereas version V4 increases it. Figure 3-ssca2 and Figure 3-vacation present the impact of this characteristic, showing that the performance is increased when we reduce the *working-set size*. As expected, version V4 presented worse performance than the original one.

## VI. Related Work

Ansari *et al.* [13] instrumented an STM implementation to collect relevant information during the execution of applications. They have showed a set of 12 metrics to characterize TM applications. They selected 3 applications from the benchmarks STAMP and Lee-TM to investigate and comprehend TM applications. Lourenço *et al.* [12] implemented a framework with low overhead, which collects transactional events and stores them in a log file. To evaluate the results, they implemented a tool to visualize the collected transactional information. Castro *et al.* [11] proposed an approach for collecting and tracing relevant information about transactions. It was based on the Linux dynamic linking mechanism, which traces events about transactions. Their solution can be applied to different STM libraries and applications since it does not modify neither the applications nor the STM source codes. Zyulkyarov *et al.* [14] proposed a series of profiling techniques for TM applications that provide comprehensive information about the wasted work caused by aborting transactions. Their study explores 3 directions: (i) identification of potential conflicts; (ii) identification of the data structures involved in conflicts; and (iii) visualization techniques to summarize how threads spend their time and which of their transactions conflict more frequently.

Our work differs from the previously mentioned ones since we focus on a general evaluation not only of applications but also of STM implementations without modifying source code and covering a larger set of transactional systems and applications. Unlike [12], [13], this approach has the advantage of using a larger variety of STM systems without modifying the source codes. Moreover, it does not add any overhead, as opposed to the works presented in [11], [14] which may modify the behavior of the applications due to extra operations to collect data.

## VII. Conclusion

This paper presented a comparative evaluation of STM systems and TM applications. We intended to answer some of the open questions in the literature and to identify opportunities for improvements in TM applications. The results of this comparative evaluation set directions and contributions that could be useful for the scientific community.

We carried out a series of experiments to better understand the performance of TM applications on state-of-the-art STM systems. Our results pointed out that there exist some important characteristics that drive the performance of TM applications. However, TM applications must be analyzed carefully to identify the most relevant characteristics that may help to improve their overall performance. As an opportunity for the future, we intend to extend this work using some tracing mechanisms as proposed in [11]. Moreover, we intend to study the impact of the TM characteristics on the performance of TM applications when executed on a real HTM processor such as the Intel Haswell.

## References

[1] T. Harris, J. Larus, and R. Rajwar, *Transactional Memory: Synthesis Lectures on Computer Architecture*, 2nd ed. Madison, USA: Morgan & Claypool Publishers, 2010, vol. 5, no. 1.

[2] M. Castro, L. F. W. Góes, L. G. Fernandes, and J.-F. Méhaut, "Dynamic Thread Mapping Based on Machine Learning for Transactional Memory Applications," in *International European Conference on Parallel and Distributed Computing (Euro-Par)*, ser. LNCS, vol. 7484. Rhodes Island, Greece: Springer-Verlag, 2012, pp. 465–476.

[3] L. F. W. Góes, C. P. Ribeiro, M. Castro, J.-F. Méhaut, M. Cole, and M. Cintra, "Automatic Skeleton-Driven Memory Affinity for Transactional Worklist Applications," *International Journal of Parallel Programming (IJPP)*, 2013.

[4] M. Hu *et. al*, "A Review of Transactional Memory in Multicore Processors," *Information Technology Journal*, vol. 9, pp. 192–200, 2010.

[5] P. Felber, C. Fetzer, and T. Riegel, "Dynamic Performance Tuning of Word-based Software Transactional Memory," in *Symposium on Principles and Practice of Parallel Programming (PPoPP)*. Salt Lake City, USA: ACM, 2008, pp. 237–246.

[6] A. Dragojević, R. Guerraoui, and M. Kapalka, "Stretching Transactional Memory," in *Programming Language Design and Implementation (PLDI)*, 2009, pp. 155–165.

[7] V. J. Marathe, M. F. Spear, C. Heriot, A. Acharya, D. Eisenstat, W. N. Scherer III, and M. L. Scott, "Lowering the Overhead of Nonbacterial Software Transactional Memory," in *ACM SIGPLAN Workshop on Transactional Computing*, Jun 2006.

[8] D. Dice *et al*, "Transactional Locking II," in *International Symposium on Distributed Computing (DISC)*, 2006, pp. 194–208.

[9] C. Minh *et al*, "STAMP: Stanford Transactional Applications for Multi-Processing," in *IEEE International Symposium on Workload Characterization (IISWC)*. Seattle, USA: IEEE Computer Society, 2008, pp. 35–46.

[10] S. Hong *et al*, "Eigenbench: A Simple Exploration Tool for Orthogonal TM Characteristics," in *IEEE International Symposium on Workload Characterization (IISWC)*. Washington, USA: IEEE Computer Society, 2010, pp. 1–11.

[11] M. Castro *et al*, "Analysis and Tracing of Applications Based on Software Transactional Memory on Multicore Architectures," in *Euromicro International Conference on Parallel, Distributed and Network-Based Computing (PDP)*. IEEE Computer Society, 2011, pp. 199–206.

[12] J. Lourenço, "Understanding the Behavior of Transactional Memory Applications," in *Workshop on Parallel and Distributed Systems: Testing, Analysis, and Debugging (PADTAD)*. Chicago, USA: ACM, 2009, pp. 3:1–3:9.

[13] M. Ansari *et al*, "Profiling Transactional Memory Applications," in *Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*. Washington, USA: IEEE Computer Society, 2009, pp. 11–20.

[14] F. Zyulkyarov *et al*, "Discovering and Understanding Performance Bottlenecks in Transactional Applications," in *International Conference on Parallel Architectures and Compilation Techniques (PACT)*. Vienna, Austria: ACM, 2010, pp. 285–294.