# RuLAM Project:
## Speculative Parallelization for Java using Software Transactional Memory*

Ivo Anjo and João Cachopo
ESW/INESC-ID Lisboa
Instituto Superior Técnico (UTL)
Rua Alves Redol 9, 1000-029 Lisboa, Portugal
{ivo.anjo,joao.cachopo}@ist.utl.pt

**ABSTRACT**

**The era of multicore processors, capable of running multiple tasks concurrently, has arrived. Sadly, most existing software and even new applications being developed are not ready to take advantage of these new multiprocessing capabilities, and, thus, more processing cores do not translate into better performance when executing these applications.**

**To tackle this problem, we envision the creation of a runtime environment that is able to parallelize automatically existing sequential applications without requiring any manual modification to those applications. To achieve this goal, we are exploring thread-level speculation supported by a Software Transactional Memory (STM), rather than relying on hardware-based mechanisms.**

**In this paper, we present our work in the RuLAM project, which targets speculative parallelization on the Java platform.**

## 1 Introduction

The transition to multicore architectures is ongoing. Modern chip designers have resigned their quest for sequential execution speed, and have shifted their resources to building multiprocessing machines that can work on many tasks concurrently.

It is only when all of the available processing cores are working that the new multicore machines fully realize their potential. Yet, most of the software we run on our computers today fails to take advantage of multicores, having little, if any, parallelism.

Regrettably, concurrent programming still presents a far greater challenge than sequential programming. Furthermore, even though new applications may take advantage of multicore architectures, a vast majority of existing code is still sequential and it is not feasible to rewrite it within a reasonable time frame. Thus, an enticing alternative is to parallelize applications automatically. In fact, there is a considerable amount of research on parallelizing compilers [1] that has produced results in the area

*This work was supported by the FCT (INESC-ID multiannual funding) through the PIDDAC Program funds and by the RuLAM project (PTDC/EIA-EIA/108240/2008).

of high performance computing for scientific applications. The problem is that parallel computing is no longer confined to the realms of these applications. The programs that we want to parallelize today require new approaches and techniques [2].

This work explores a more dynamic approach — speculative parallelization — in which parallelization decisions are optimistic. There have been other proposals for both hardware [3, 4] and software-based [5, 6] speculative parallelization systems. We propose a system supported by Software Transactional Memory (STM), which provides a model similar to database transactions, but over memory read and write operations. Under STM, a thread can begin a transaction, then proceed to operate in isolation and finally it can atomically apply its changes at the end of the transaction, after successful validation. We argue that such a system has several advantages over hardware-supported approaches. First, because STM-based executions may be unbounded, we can extend the range of possible parallelizations, thereby increasing the potential for extracting parallelism from applications. Second, we may apply these techniques to applications that run on machines that do not support transactional execution, including all of the current mainstream hardware. Finally, we may leverage on much of the intense research being done in the area of transactional memory.

The differentiating feature of our approach is that we are targeting long-running speculative executions, that may span multiple method calls, spawn other speculative executions, and be able to execute non-transactional operations such as I/O without aborting and discarding potentially useful work.

To achieve our targets, we have been working on modifications to the JVSTM software transactional memory, and on implementing a speculative parallelization framework to support our research — the JaSPEx framework.

The remainder of this short paper is organized as follows. In Section 2, we discuss the usage of software transactional memory for speculative parallelization. Then, in Section 3, we present the JaSPEx framework, and, finally, in Section 4, we summarize the current findings and future work.

## 2 Speculative parallelization based on STM

In the RuLAM project, our first prototypes were based on the Java Versioned Software Transactional Memory (JVSTM) [7, 8], which is a pure Java STM library. The JVSTM is a versioned STM: memory positions keep not only the latest values committed to them, but also the history of older values written in the past. This arrangement allows read-only transactions to never conflict with any other concurrent transaction, favoring applications that have a high read/write transaction ratio.

Yet, using an STM for supporting thread-level speculation (TLS) [2] provides a very different use-case from normal STM usage. In particular, issues such as commit ordering, the atomicity model provided, and the nesting model supported present opportunities for optimization. To take advantage of this fact, we are creating a modified version of the JVSTM that is optimized for TLS usage, and that can be used as a drop-in replacement for the regular JVSTM in JaSPEx, our parallelization research framework.

One of our goals is to allow speculative executions to go beyond what most current TLS systems allow. This includes the possibility of speculative executions spawning other speculative executions, and so we are also working on replacing the current *linear nesting* model on the JVSTM, where concurrency within transactions is disallowed, with one that allows parallelism when using nested transactions, so that a hierarchy of parent, sibling and children transactions inside a single top-level transaction may run concurrently.

Because the JVSTM is a normal Java library, applications must first be modified to behave transactionally under its control. This functionality is currently provided by the transactification part of the JaSPEx framework. Moreover, because a pure-library STM imposes certain limitations and overheads, we are investigating the possibility of implementing the STM at the Java Virtual Machine (JVM) level, to improve performance and allow better cooperation with the JVM.

# 3 The JaSPEx parallelization research framework

We have created the Java Speculative Parallel Executor (JaSPEx) framework as a testing ground for software TLS parallelization techniques. It is implemented as a Java library that can run on any Java Virtual Machine, and provides a custom class loader that intercepts the loading of Java classes in bytecode form, and modifies them to execute speculatively in parallel.

Because the JaSPEx framework works directly on the application bytecode, it can be applied to any language that can be compiled to run on the JVM – not just Java – and it does not need access to the original source code.

## 3.1 Transactification of Applications

Before applying any further changes, applications are modified to run transactionally under the control of the STM, a step we call the "transactification" of the application. This step involves intercepting and modifying accesses to class and instance fields, to arrays (for which currently there is only limited support), and handling of non-transactional operations, such as calls to native code, I/O operations, and other operations that cannot be undone by the STM.

Two strategies are supported for handling non-transactional operations that occur during a speculative execution: the speculative execution can be aborted, or it can be dynamically transitioned to a non-speculative execution after validation, allowing the already accomplished work to be merged with the global application state.

## 3.2 Speculative Execution

After transactification, the framework identifies the tasks that are candidates for speculative execution. Multiple task identification strategies can be used for this step.

Whenever code execution reaches a possible task spawn point, a call is made into the framework, allowing it to decide if the execution should proceed in parallel or not, a decision that may also be influenced by the task scheduler.

If the framework decides to proceed with the speculative execution, one or more tasks are created and queued for execution by an `ExecutorService`, which keeps a pool of worker threads. Tasks wait on other tasks for their results in a scheme similar to fork/join [9] parallelism, but our framework differs from other fork/join-like approaches by being automatic and not needing programmer input.

## 3.3 Task Scheduling

Also being developed is a scheduler that supports multiple strategies for task scheduling. Our framework-level scheduler combines static code information with past speculation statistics to try to minimize speculation conflicts and make better use of the available machine resources.

# 4 Conclusions and Future work

In this paper, we gave a brief presentation of the RuLAM project, its objectives, and our current work. We argued that an automatic parallelization approach should be based on an STM, and we briefly described some of the aspects that must be taken into account by the STM, if it is to be used to support speculative parallelization.

We also presented JaSPEx, a software TLS framework that we created to experiment with parallelization techniques. We described some of its functions, such as transactifying an application and speculative task selection, creation, and scheduling.

Preliminary results show that some benchmarks can benefit from our approach, but our implementation has to be further fine-tuned to reduce execution overheads. As the current JaSPEx framework works on an unmodified JVM, we plan to investigate the effects of moving some of its functions inside the JVM itself.

Finally, we hope that our approach will allow us to uncover new, untapped parallelism in common applications. We also hope to gain insights on why some applications cannot be parallelized, and how they could be changed to benefit from parallelization.

# References

[1] B. Blume, R. Eigenmann, K. Faigin, J. Grout, J. Hoeflinger, D. Padua, P. Petersen, B. Pottenger, L. Rauchwerger, P. Tu, et al. Polaris: The Next Generation in Parallelizing Compilers. In Proceedings of the Seventh Workshop on Languages and Compilers for Parallel Computing, 1994.

[2] J.T. Oplinger, D.L. Heine, and M.S. Lam. In search of speculative thread-level parallelism. In 1999 International Conference on Parallel Architectures and Compilation Techniques, 1999. Proceedings, pages 303–313, 1999.

[3] W. Liu, J. Tuck, L. Ceze, W. Ahn, K. Strauss, J. Renau, and J. Torrellas. POSH: a TLS compiler that exploits program structure. In PPoPP '06: Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming, pages 158–167, New York, NY, USA, 2006. ACM.

[4] M.K. Chen and K. Olukotun. The Jrpm system for dynamically parallelizing Java programs. In Proceedings of the 30th annual international symposium on Computer architecture, pages 434–446. ACM New York, NY, USA, 2003.

[5] C. Pickett and C. Verbrugge. Software thread level speculation for the Java language and virtual machine environment. Languages and Compilers for Parallel Computing, pages 304–318, 2006.

[6] M.F. Spear, K. Kelsey, T. Bai, L. Dalessandro, M.L. Scott, C. Ding, and P. Wu. Fastpath speculative parallelization. In Proceedings of the Workshop on Languages and Compilers for Parallel Computing. Springer, 2009.

[7] J. Cachopo and A. Rito-Silva. Versioned boxes as the basis for memory transactions. Science of Computer Programming, 63(2):172–185, 2006.

[8] J. Cachopo. JVSTM - Java Versioned Software Transactional Memory, 2008. `http://web.ist.utl.pt/joao.cachopo/jvstm/`.

[9] D. Lea. A Java fork/join framework. In Proceedings of the ACM 2000 conference on Java Grande, pages 36–43. ACM New York, NY, USA, 2000.