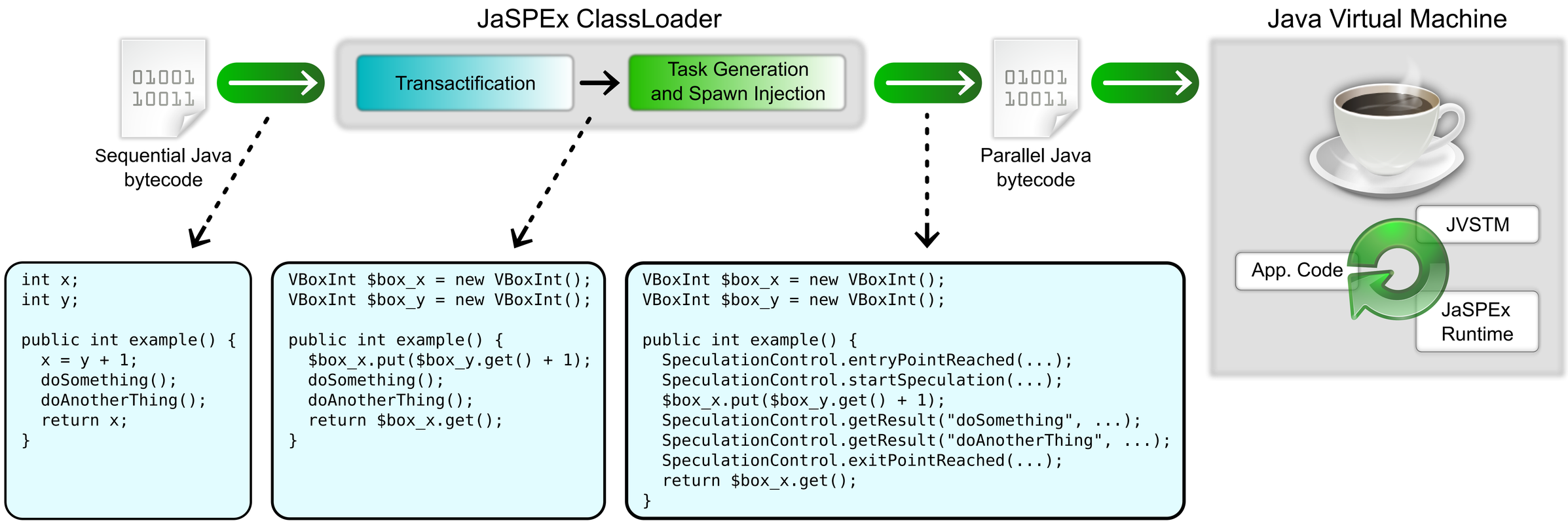
RuLAM Project: Speculative Parallelization for Java using Software Transactional Memory

Ivo Anjo ivo.anjo@ist.utl.pt João Cachopo joao.cachopo@ist.utl.pt

ESW / INESC-ID Lisboa / Instituto Superior Técnico / Universidade Técnica de Lisboa



How can we take advantage of modern multicore machines to make applications run faster?

Problem Statement and Goals

- Legacy sequential applications do not benefit from multicores
- Hard to rewrite or adapt them

We propose to solve this issue by creating a parallelization system that does thread-level speculation (TLS) based on Software Transactional Memory (STM).

Why use STM?

- Executions may be unbounded, spanning multiple method calls
- We can target mainstream hardware with no built-in transactional support
- We can leverage on current STM research

We are aiming to support:

- **Long-running** speculative executions
- Executions that may span **multiple method calls** and include **nesting**
- **Dynamic transitions** between speculative execution and *program-order* for executing non-transactional operations

Speculative Execution based on STM

Doing thread-level speculation using an STM differs from normal STM usage:

- Transactions will commit in an order that can be known ahead-of-time
- Atomicity model needs to be considered when a non-speculative thread interacts with threads executing speculative tasks
- Nesting allows speculative executions to spawn other speculative executions

JaSPEx Parallelization Framework

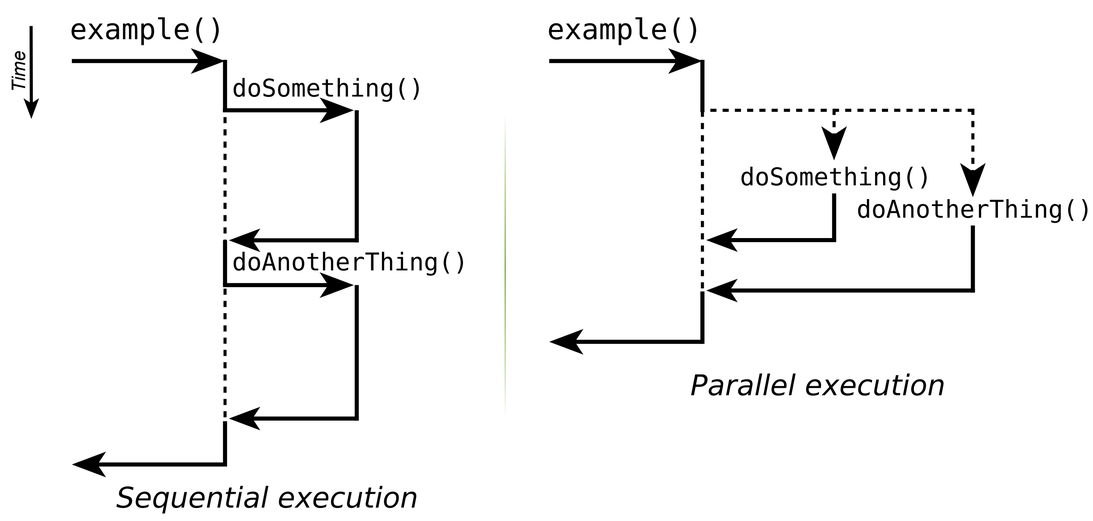
The Java Speculative Parallel Executor (JaSPEx) framework was created as a testing ground for software TLS parallelization that:

- Works directly on **bytecode**, and is implemented as a **Java library**
- Provides a `ClassLoader` that modifies applications as they are loaded

Transactification

Application code must first pass through a *transactification* step, where it is modified to let the STM/JaSPEx handle:

- Accesses to class and instance fields
- Array operations
- Non-transactional operations (I/O, native, ...)



Speculation

Application code is analyzed and modified as it is loaded:

- Methods that *look like* they can be run in parallel are identified
- Hooks are installed to trigger the queuing of speculative tasks for execution

At runtime:

- When execution reaches a given spawn point, one or more tasks are created
- Tasks are picked up and executed by a pool of threads

The current task commit and waiting scheme is similar to the fork/join approach.

Task Scheduling

- It is common to have more tasks than available processors
- Some tasks might cause conflicts whenever they are executed in parallel

To tackle these issues, we are working on a scheduler that combines static information with past speculation results.

Open Issues and Conclusion

- Can a better STM reduce the overheads of *transactification*, or even eliminate them for the thread running in *program-order*?
- How does fork/join style speculation compares to call continuation speculation?
- What is the right balance between task size and the overheads for speculative execution?

We hope our approach will allow us to uncover new, untapped parallelism in common applications; or otherwise identify why not and how can an application be modified to benefit from our approach.

Acknowledgements

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).

INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

inesc id lisboa