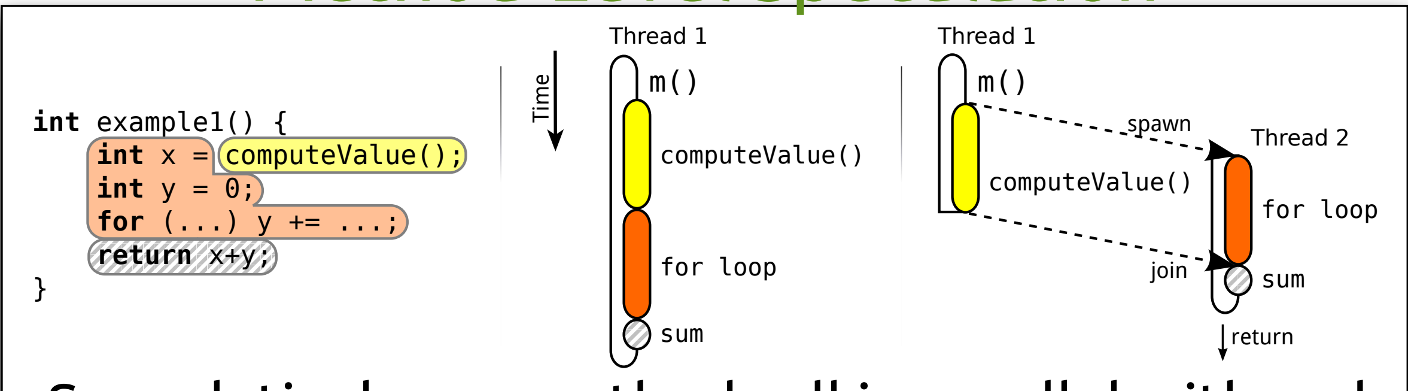
Speculative Parallelization of Object-Oriented JVM Applications

Ivo Anjo `ivo.anjo@ist.utl.pt` Advisor: João Cachopo

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

Based On

Method-Level Speculation


```
int example1() {
    int x = computeValue();
    int y = 0;
    for (...) y += ...;
    return x+y;
}
```

- Speculatively run method call in parallel with code following its return

OpenJDK

Based on modified version of OpenJDK:
- Support for first-class continuations
- Support for JDK transactification
- Production-class performance

Software Transactional Memory

Custom transactional model for speculative parallelization:
- Value-based
- Low overhead for program-order thread

... with extensions

Bytecode Preparation

Transactification

```
object.field = 100l;
```
↓
```
TM.storeLong(object, field, 100l);
```

Non-Transactional Operations

Added before every non-transactional operation

```
nonTransactionalActionAttempted();
System.out.println("Hello, world!");
```

Futures in the Write-Set

```
void doCompute(Object[] res) {
    for (int i = 0; i < res.length; i++) {
        res[i] = compute(i);
    }}
```
↘
```
void doCompute(Object[] res) {
    for (int i = 0; i < res.length; i++) {
        Future temp1 = spawn compute(i);
        TM.storeFutureArray(res, i, temp1);
            // temp1 is handed to the STM
    }}
```

Spawn and Future Insertion

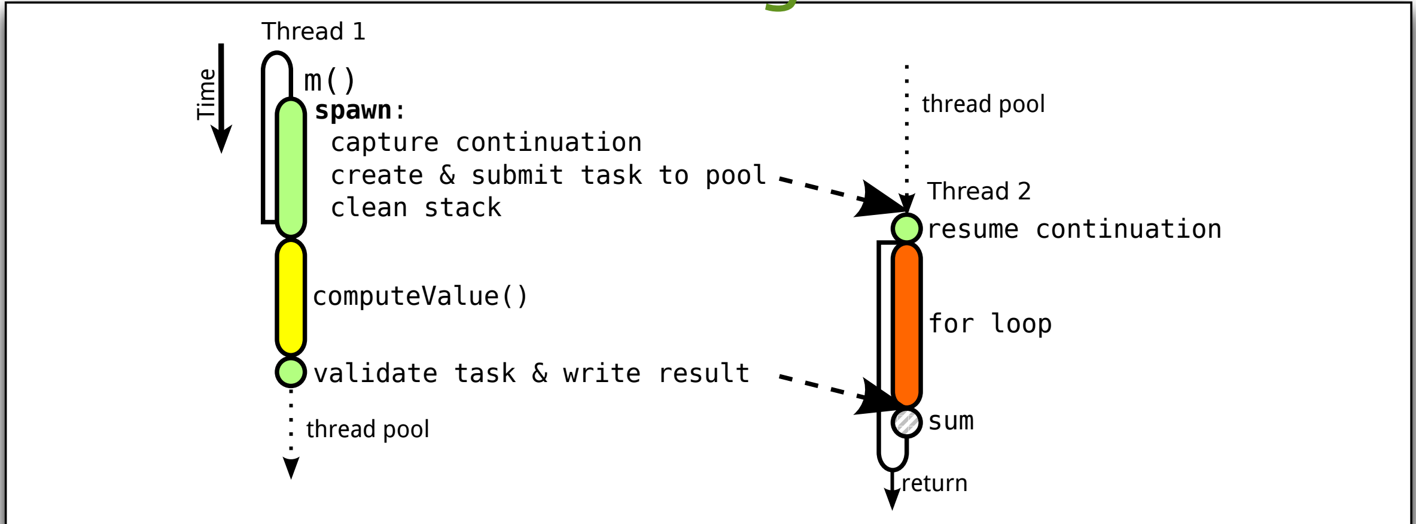
```
int example() {
    return compute(0) + compute(1);
}
```
→
```
int example() {
    Future temp1 = spawn compute(0);
    int temp2 = compute(1);
    return temp1.get() + temp2;
}
```

JaSPEx Framework

Software speculative parallelization for Java irregular applications (OO/method-heavy)

Runtime Features

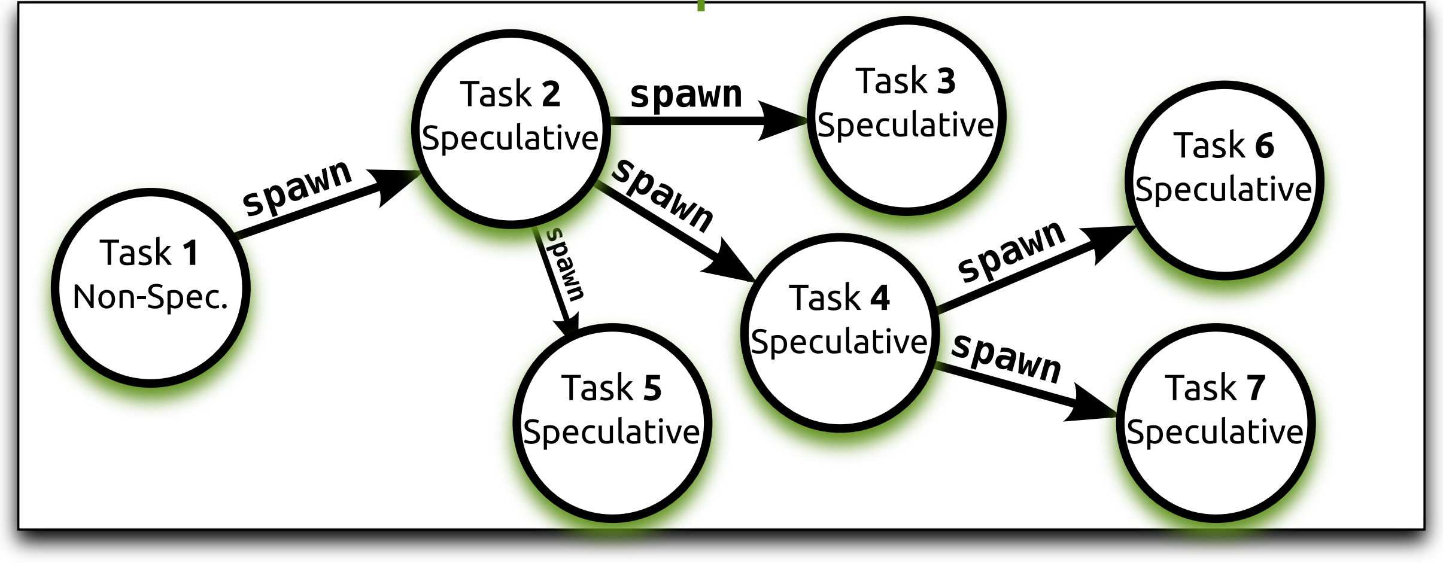
State Transfer using Continuations



Return Value Prediction

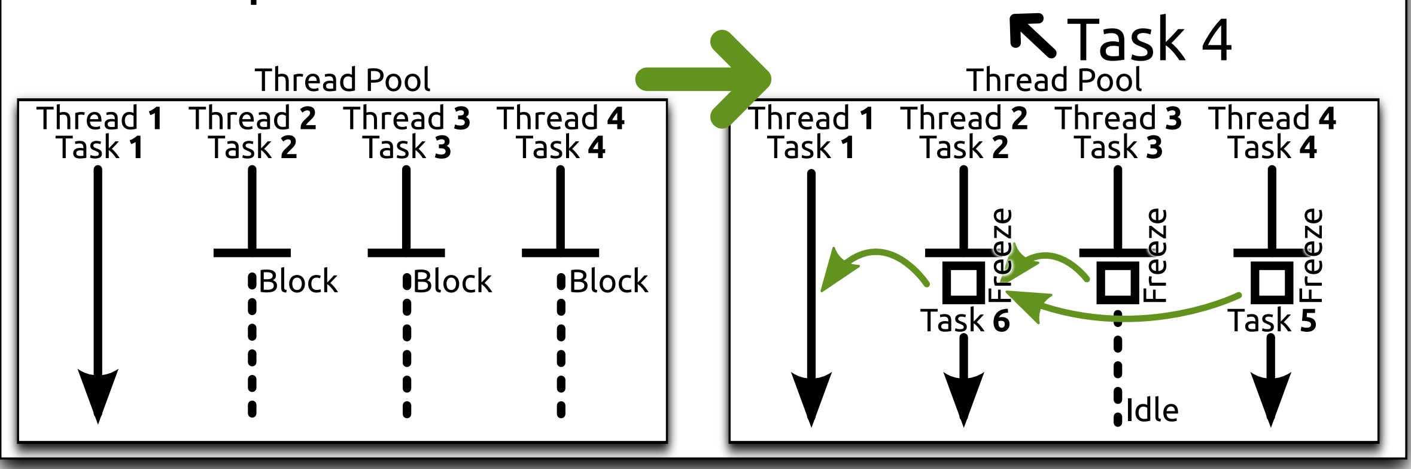
```
void example(int val) {
    Future i = spawn compute(0);
    int j = val + i.get();
    ...
}
```
Instead of blocking:
- Predict expected value
- Register read of value with STM

Nested Speculation



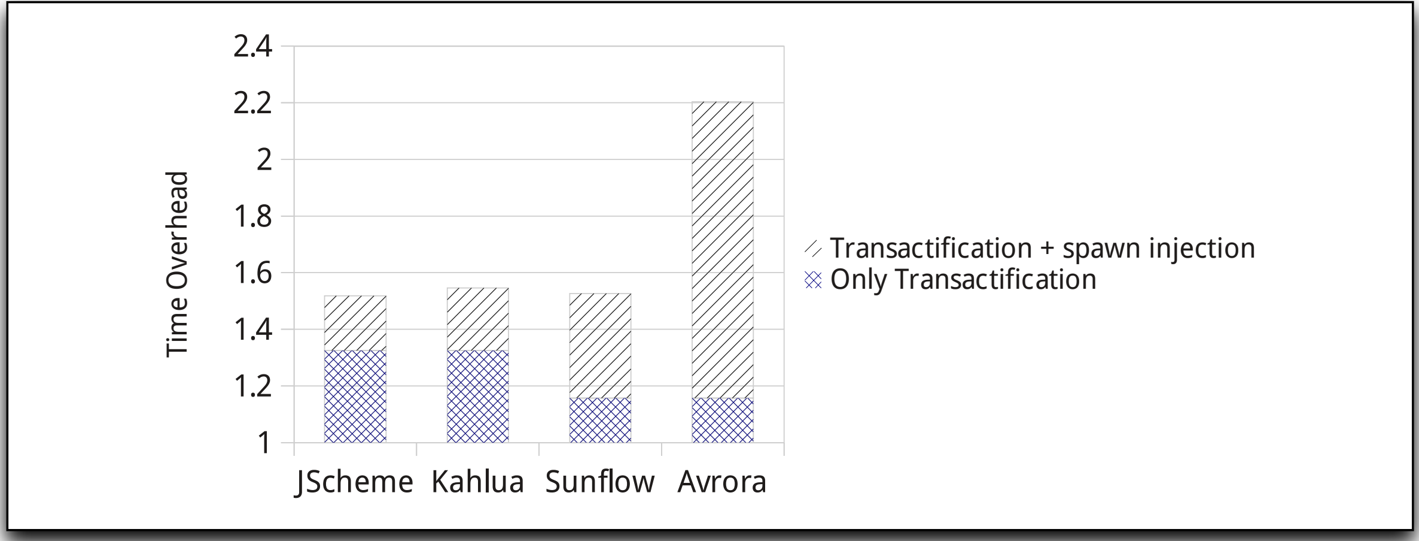
Task Freeze and Thread Reuse

Dependences: Task 1 ← Task 2 ← Task 3
↖ Task 4

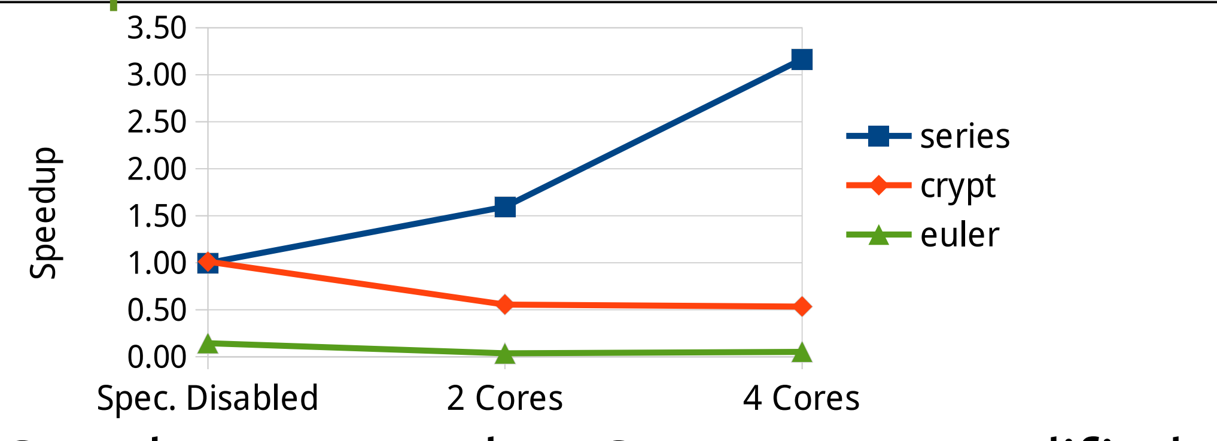


Results

Low Overhead for Transformations



Speculative Parallelization



Java Grande Framework — Sources **not** modified

Intel Core i5 750 / Ubuntu Linux 12.10 64-Bit

Open Issues and Conclusions

- I still need better tools for profiling, avoiding unuseful tasks
- How to modify the JDK for transactional use

- JaSPEx-MLS is able to uncover untapped parallelism in applications

Acknowledgements

This work was supported by national funds through FCT – Fundação para a Ciência e a Tecnologia, both under project PEst-OE/EEI/LA0021/2013 and under project PTDC/EIA-EIA/108240/2008 (the RuLAM project).

INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

inesc id lisboa