JaSPEx: Speculative Parallel Execution of Java Applications

Ivo Anjo   João Cachopo

ESW
INESC-ID Lisboa/Instituto Superior Técnico

Setembro de 2009

Multicore Processors

Multicore Processors

- Are the future

Multicore Processors

- Are the future
- Are only fully used when all cores have work

Multicore Processors

- Are the future
- Are only fully used when all cores have work
- New processors speed up all applications

Multicore Processors

- Are the future
- Are only fully used when all cores have work
- ~~New processors speed up all applications~~
  - Most existing code is not parallel

Problem

Problem

- New processors do not speed up legacy applications

Problem

- New processors do not speed up legacy applications
- Problematic to rewrite or adapt these applications

Problem

- New processors do not speed up legacy applications
- Problematic to rewrite or adapt these applications

Solution

Problem

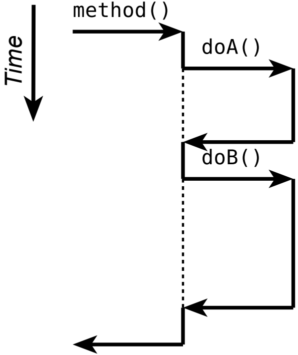
- New processors do not speed up legacy applications
- Problematic to rewrite or adapt these applications

Solution

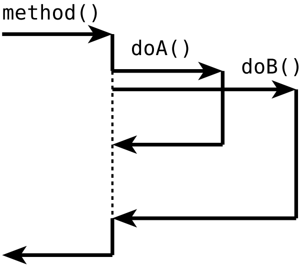
- Automatic Parallelization

```java
void method() {
    doA();
    doB();
}
```

```java
void method() {
    doA();
    doB();
}
```



Sequential execution



Parallel execution

Problem

Problem

- Can `doA()` and `doB()` be run in parallel?

Problem

- Can `doA()` and `doB()` be run in parallel?
    - Shared state

Problem

- Can `doA()` and `doB()` be run in parallel?

Parallelizing Compilers

Problem

- Can `doA()` and `doB()` be run in parallel?

Parallelizing Compilers

- Only if we can prove that they can never interfere

Problem

- Can `doA()` and `doB()` be run in parallel?

Parallelizing Compilers

- Only if we can prove that they can never interfere

Speculative Parallelization

Problem

- Can `doA()` and `doB()` be run in parallel?

Parallelizing Compilers

- Only if we can prove that they can never interfere

Speculative Parallelization

- Maybe, let’s try and we’ll see how it goes

Speculative Parallelization

a.k.a. Thread-Level Speculation (TLS)

Speculative Parallelization

a.k.a. Thread-Level Speculation (TLS)

- Parallelization system does not have to prove that a parallelization is valid

Speculative Parallelization

a.k.a. Thread-Level Speculation (TLS)

- Parallelization system does not have to prove that a parallelization is valid
- Uses memory transactions, that are aborted and undone if the original execution semantics are violated

Speculative Parallelization

a.k.a. Thread-Level Speculation (TLS)

Speculative Parallelization

a.k.a. Thread-Level Speculation (TLS)

- Most TLS approaches need hardware support for transactions (HTM)

Speculative Parallelization

a.k.a. Thread-Level Speculation (TLS)

- Most TLS approaches need hardware support for transactions (HTM)
  - Limits duration

Speculative Parallelization

a.k.a. Thread-Level Speculation (TLS)

- Most TLS approaches need hardware support for transactions (HTM)
    - Limits duration
    - Limits size

Speculative Parallelization

a.k.a. Thread-Level Speculation (TLS)

- Most TLS approaches need hardware support for transactions (HTM)
    - Limits duration
    - Limits size
    - Do not work on my new PC!!

Speculative Parallelization

a.k.a. Thread-Level Speculation (TLS)

- Most TLS approaches need hardware support for transactions (HTM)
    - Limits duration
    - Limits size
    - Mainstream architectures offer no such support

Speculative Parallelization

a.k.a. Thread-Level Speculation (TLS)

- Most TLS approaches need hardware support for transactions (HTM)
  - Limits duration
  - Limits size
  - Mainstream architectures offer no such support

- Our proposal: TLS using a Software Transactional Memory (STM)

JaSPEx

JaSPEx

Java Speculative Parallel Executor

JaSPEx

Java Speculative Parallel Executor

- TLS system for Java/JVM

JaSPEx

Java Speculative Parallel Executor

- TLS system for Java/JVM
- Works at the JVM bytecode level

JaSPEx

Java Speculative Parallel Executor

- TLS system for Java/JVM
- Works at the JVM bytecode level
- Uses the JVSTM

JaSPEx

Java Speculative Parallel Executor

- TLS system for Java/JVM
- Works at the JVM bytecode level
- Uses the JVSTM
  - Java Versioned Software Transactional Memory

JaSPEx

Java Speculative Parallel Executor

- TLS system for Java/JVM
- Works at the JVM bytecode level
- Uses the JVSTM
  - Java Versioned Software Transactional Memory
  - STM implemented in Java

JaSPEx

Java Speculative Parallel Executor

- TLS system for Java/JVM
- Works at the JVM bytecode level
- Uses the JVSTM
  - Java Versioned Software Transactional Memory
  - STM implemented in Java
- Also done in Java

JaSPEx

Static Modification Module

JaSPEx

Static Modification Module

Runtime Control Module

JaSPEx

Static Modification Module

- `ClassLoader` that rewrites bytecode

Runtime Control Module

JaSPEx

Static Modification Module

- `ClassLoader` that rewrites bytecode
- *Transactification*

Runtime Control Module

JaSPEx

Static Modification Module

- `ClassLoader` that rewrites bytecode
- *Transactification*
- Prevention of nontransactional operations

Runtime Control Module

JaSPEx

Static Modification Module

- `ClassLoader` that rewrites bytecode
- *Transactification*
- Prevention of nontransactional operations

Runtime Control Module

- Controls speculation

Transactification

Transactification

- No support for memory transactions on the JVM

Transactification

- No support for memory transactions on the JVM
- JVSTM is just a Java library

Transactification

- No support for memory transactions on the JVM
- JVSTM is just a Java library

- Application needs to be modified to use the JVSTM

Transactification

- No support for memory transactions on the JVM
- JVSTM is just a Java library

- Application needs to be modified to use the JVSTM
  ⇒ "Transactification"

JVSTM Crash-course

JVSTM Crash-course

jvstm.VBox<E>

JVSTM Crash-course

jvstm.VBox<E>

jvstm.Transaction

JVSTM Crash-course

jvstm.VBox<E>

- Container that keeps the version history of a memory position

jvstm.Transaction

JVSTM Crash-course

`jvstm.VBox<E>`

- Container that keeps the version history of a memory position
    - Fields (class or instance)

`jvstm.Transaction`

JVSTM Crash-course

`jvstm.VBox<E>`

- Container that keeps the version history of a memory position
  - Fields (class or instance)
  - Array positions

`jvstm.Transaction`

JVSTM Crash-course

`jvstm.VBox<E>`

- Container that keeps the version history of a memory position
  - Fields (class or instance)
  - Array positions
  - Local variables

`jvstm.Transaction`

JVSTM Crash-course

`jvstm.VBox<E>`

- Container that keeps the version history of a memory position
- `E get()`

`jvstm.Transaction`

JVSTM Crash-course

`jvstm.VBox<E>`

- Container that keeps the version history of a memory position
- `E get()`
- `void put(E newE)`

`jvstm.Transaction`

JVSTM Crash-course

`jvstm.VBox<E>`

- Container that keeps the version history of a memory position
- `E get()`
- `void put(E newE)`

`jvstm.Transaction`

- `Transaction begin()`

JVSTM Crash-course

`jvstm.VBox<E>`

- Container that keeps the version history of a memory position
- `E get()`
- `void put(E newE)`

`jvstm.Transaction`

- `Transaction begin()`
- `void commit()`

JVSTM Crash-course

`jvstm.VBox<E>`

- Container that keeps the version history of a memory position
- `E get()`
- `void put(E newE)`

`jvstm.Transaction`

- `Transaction begin()`
- `void commit()`
- `void abort()`

Transactification

Transactification

```
public class Person {
    public String name;

    public Person(String name) {
        this.name = name;
    }

    public String name() {
        return name;
    }
}
```

Transactification

```
public class Person {
    public VBox<String> $box_name;

    public Person(String name) {
        this.name = name;
    }

    public String name() {
        return name;
    }
}
```

- Replace the original fields with VBoxes

Transactification

```
public class Person {
    private VBox<String> $box_name;

    public Person(String name) {
        this.name = name;
    }

    public String name() {
        return name;
    }
}
```

- Replace the original fields with VBoxes

Transactification

```
public class Person {
    private VBox<String> $box_name = new VBox<String>();

    public Person(String name) {
        this.name = name;
    }

    public String name() {
        return name;
    }
}
```

- Add VBox initializations

Transactification

```
public class Person {
    private VBox<String> $box_name = new VBox<String>();

    public Person(String name) {
        this.name = name;
    }

    public String name() {
        return name;
    }

    String $box_name_get() { return $box_name.get(); }
    void $box_name_put(String name) { $box_name.put(name); }
}
```

- Add get and put methods, that control access to the VBox

Transactification

```
public class Person {
    private VBox<String> $box_name = new VBox<String>();

    public Person(String name) {
        this.name = name;
    }

    public String name() {
        return name;
    }

    public String $box_name_get() { return $box_name.get(); }
    public void $box_name_put(String name) { $box_name.put(name); }
}
```

- Add `get` and `put` methods, that control access to the `VBox`

Transactification

```
public class Person {
    private VBox<String> $box_name = new VBox<String>();

    public Person(String name) {
        $box_name_put(name);
    }

    public String name() {
        return $box_name_get();
    }

    public String $box_name_get() { return $box_name.get(); }
    public void $box_name_put(String name) { $box_name.put(name); }
}
```

- Replace accesses to the original fields with calls to the get and put methods

Transactification Problems

Transactification Problems

- `native methods`

Transactification Problems

- `native` methods
- The Sun JVM does not allow loading modified versions of classes in the `java.*` package

Transactification Problems

- `native` methods
- The Sun JVM does not allow loading modified versions of classes in the `java.*` package
- ...

Transactification Problems

- `native` methods
- The Sun JVM does not allow loading modified versions of classes in the `java.*` package
- ...

⇒ Nontransactional operations need to be detected and handled

Prevention of nontransactional operations

Prevention of nontransactional operations

Idea

Prevention of nontransactional operations

Idea

- Before executing each nontransactional operation, add a call to the framework

Prevention of nontransactional operations

Idea

- Before executing each nontransactional operation, add a call to the framework

```
...
System.out.println("Hello World!");
...
```

Prevention of nontransactional operations

Idea

- Before executing each nontransactional operation, add a call to the framework

```
...
SpeculationControl.nonTransactionalActionAttempted();
System.out.println("Hello World!");
...
```

Speculation

Speculation

- Speculative execution of methods

Speculation

- Speculative execution of methods
- When a method is invoked, some of the methods it invokes may be run speculatively

```java
public static int fib(int n) {
    if (n <= 1) return n;
    return fib(n-1) + fib(n-2);
}
```

```
public static int fib(int n) {
    if (n <= 1) return n;
    return fib(n-1) + fib(n-2);
}
```

```
public static int fib(int n) {
```

```
public static int fib(int n) {
    if (n <= 1) return n;
    return fib(n-1) + fib(n-2);
}
```

```
public static int fib(int n) {
    SpeculationId specId =
        SpeculationControl.entryPointReached(ENTRY_POINT_ID,
public static int fib(int n) {
    if (n <= 1) return n;
    return fib(n-1) + fib(n-2);
}
```

```
public static int fib(int n) {
    SpeculationId specId =
        SpeculationControl.entryPointReached(ENTRY_POINT_ID,
            new Object[] { new Object[] { n-1 },
                           new Object[] { n-2 } });
```

```java
public static int fib(int n) {
    if (n <= 1) return n;
    return fib(n-1) + fib(n-2);
}
```

```java
public static int fib(int n) {
    SpeculationId specId =
        SpeculationControl.entryPointReached(ENTRY_POINT_ID,
            new Object[] { new Object[] { n-1 },
                           new Object[] { n-2 } });
    if (n <= 1) {
```

```
public static int fib(int n) {
    if (n <= 1) return n;
    return fib(n-1) + fib(n-2);
}
```

```
public static int fib(int n) {
    SpeculationId specId =
        SpeculationControl.entryPointReached(ENTRY_POINT_ID,
            new Object[] { new Object[] { n-1 },
                           new Object[] { n-2 } });
    if (n <= 1) {
        SpeculationControl.exitPointReached(specId);
```

```java
public static int fib(int n) {
    if (n <= 1) return n;
    return fib(n-1) + fib(n-2);
}
```

```java
public static int fib(int n) {
    SpeculationId specId =
        SpeculationControl.entryPointReached(ENTRY_POINT_ID,
            new Object[] { new Object[] { n-1 },
                           new Object[] { n-2 } });
    if (n <= 1) {
        SpeculationControl.exitPointReached(specId);
        return n;
    }
```

```
public static int fib(int n) {
    if (n <= 1) return n;
    return fib(n-1) + fib(n-2);
}
```

```
public static int fib(int n) {
    SpeculationId specId =
        SpeculationControl.entryPointReached(ENTRY_POINT_ID,
            new Object[] { new Object[] { n-1 },
                           new Object[] { n-2 } });
    if (n <= 1) {
        SpeculationControl.exitPointReached(specId);
        return n;
    }
    Future f0 = SpeculationControl.getResult(specId, ID0);
```

```
public static int fib(int n) {
    if (n <= 1) return n;
    return fib(n-1) + fib(n-2);
}
```

```
public static int fib(int n) {
    SpeculationId specId =
        SpeculationControl.entryPointReached(ENTRY_POINT_ID,
            new Object[] { new Object[] { n-1 },
                           new Object[] { n-2 } });
    if (n <= 1) {
        SpeculationControl.exitPointReached(specId);
        return n;
    }
    Future f0 = SpeculationControl.getResult(specId, ID0);
    Future f1 = SpeculationControl.getResult(specId, ID1);
```

```
public static int fib(int n) {
    if (n <= 1) return n;
    return fib(n-1) + fib(n-2);
}
```

```
public static int fib(int n) {
    SpeculationId specId =
        SpeculationControl.entryPointReached(ENTRY_POINT_ID,
            new Object[] { new Object[] { n-1 },
                           new Object[] { n-2 } });
    if (n <= 1) {
        SpeculationControl.exitPointReached(specId);
        return n;
    }
    Future f0 = SpeculationControl.getResult(specId, ID0);
    Future f1 = SpeculationControl.getResult(specId, ID1);
    int temp = f0.get() + f1.get();
```

```java
public static int fib(int n) {
    if (n <= 1) return n;
    return fib(n-1) + fib(n-2);
}
```

```java
public static int fib(int n) {
    SpeculationId specId =
        SpeculationControl.entryPointReached(ENTRY_POINT_ID,
            new Object[] { new Object[] { n-1 },
                           new Object[] { n-2 } });
    if (n <= 1) {
        SpeculationControl.exitPointReached(specId);
        return n;
    }
    Future f0 = SpeculationControl.getResult(specId, ID0);
    Future f1 = SpeculationControl.getResult(specId, ID1);
    int temp = f0.get() + f1.get();
    SpeculationControl.exitPointReached(specId);
```

```
public static int fib(int n) {
    if (n <= 1) return n;
    return fib(n-1) + fib(n-2);
}
```

```
public static int fib(int n) {
    SpeculationId specId =
        SpeculationControl.entryPointReached(ENTRY_POINT_ID,
            new Object[] { new Object[] { n-1 },
                           new Object[] { n-2 } });
    if (n <= 1) {
        SpeculationControl.exitPointReached(specId);
        return n;
    }
    Future f0 = SpeculationControl.getResult(specId, ID0);
    Future f1 = SpeculationControl.getResult(specId, ID1);
    int temp = f0.get() + f1.get();
    SpeculationControl.exitPointReached(specId);
    return temp;
}
```

Doing Speculation

Doing Speculation

- Call to `entryPointReached(...)` generates tasks

Doing Speculation

- Call to `entryPointReached(...)` generates tasks
    - `fib(n-1)`

Doing Speculation

- Call to `entryPointReached(...)` generates tasks
    - `fib(n-1)`
    - `fib(n-2)`

Doing Speculation

- Call to `entryPointReached(...)` generates tasks
  - `fib(n-1)`
  - `fib(n-2)`

- Each task is executed inside an STM transaction

Transaction Commit

Transaction Commit

- Tasks wait permission for commit

Transaction Commit

- Tasks wait permission for commit
  - Execute nontransactional operations

Transaction Commit

- Tasks wait permission for commit
    - Execute nontransactional operations
    - Termination

Transaction Commit

- Tasks wait permission for commit
    - Execute nontransactional operations
    - Termination
    - Join with child speculation task

Transaction Commit

- Tasks wait permission for commit
    - Execute nontransactional operations
    - Termination
    - Join with child speculation task
- Commit needs to guarantee the original semantics

Transaction Commit

- Tasks wait permission for commit
  - Execute nontransactional operations
  - Termination
  - Join with child speculation task
- Commit needs to guarantee the original semantics

Transaction Commit

- Tasks wait permission for commit
  - Execute nontransactional operations
  - Termination
  - Join with child speculation task

- Commit needs to guarantee the original semantics
  ⇒ Threads commit in program-order

Transaction Commit

- Tasks wait permission for commit
  - Execute nontransactional operations
  - Termination
  - Join with child speculation task

- Commit needs to guarantee the original semantics
  $\Rightarrow$ Threads commit in program-order

- At each moment, only one thread is running in program-order, and it can yield the program-order to other tasks

Transaction Commit

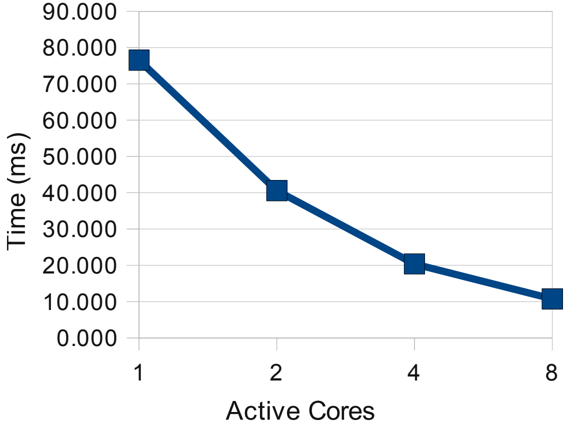
- Tasks wait permission for commit
  - Execute nontransactional operations
  - Termination
  - Join with child speculation task

- Commit needs to guarantee the original semantics
  $\Rightarrow$ Threads commit in program-order

- At each moment, only one thread is running in program-order, and it can yield the program-order to other tasks
  - Which can then try to commit

Preliminary Experimental Results

Preliminary Experimental Results



Time for calculating `fib(50)` with 1..8 cores.
A fib version modified with a threshold was used.

Conclusions

Conclusions

- *Transactification* of Java/JVM applications

Conclusions

- *Transactification* of Java/JVM applications
- *Transactification* without support from the JVM

Conclusions

- *Transactification* of Java/JVM applications
- *Transactification* without support from the JVM
    - Hard

Conclusions

- *Transactification* of Java/JVM applications
- *Transactification* without support from the JVM
  - Hard
  - Imposes limitations

Conclusions

- *Transactification* of Java/JVM applications
- *Transactification* without support from the JVM
  - Hard
  - Imposes limitations
- It is possible to achieve speedups for some applications

Future Work

Future Work

- Realistic benchmarks

Future Work

- Realistic benchmarks
- Reducing overheads

Future Work

- Realistic benchmarks
- Reducing overheads
    - Optimize the JVSTM for our use-case

Future Work

- Realistic benchmarks
- Reducing overheads
  - Optimize the JVSTM for our use-case
- Gather statistics

Future Work

- Realistic benchmarks
- Reducing overheads
  - Optimize the JVSTM for our use-case
- Gather statistics

- Support *transactification* at the JVM-runtime level

Thank you!

Questions?