A Software-based Method-Level Speculation Framework for the Java Platform

Ivo Anjo João Cachopo

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

September 2012

IBM POWER4 (**2C**)

Intel Pentium D (**2C**)
AMD Athlon 64 X2 (**2C**)
Sun UltraSPARC T1 (**8C**)
Azul Vega 1 (**24C**)

IBM Xenon 'X360' (**3C**)

Intel Xeon 'Beckton' (**8C**)
AMD Opteron 'Magny-Cours' (**12C**)

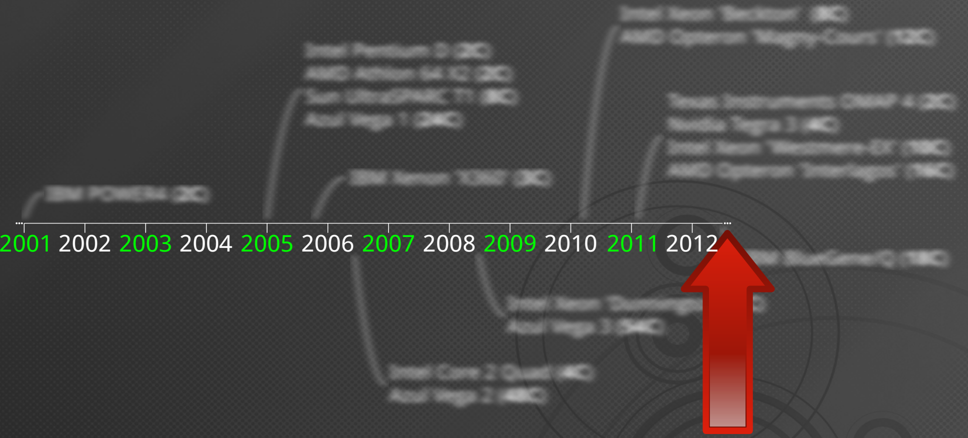
Texas Instruments OMAP 4 (**2C**)
Nvidia Tegra 3 (**4C**)
Intel Xeon 'Westmere-EX' (**10C**)
AMD Opteron 'Interlagos' (**16C**)

2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012

IBM BlueGene/Q (**18C**)

Intel Xeon 'Dunnington' (**6C**)
Azul Vega 3 (**54C**)

Intel Core 2 Quad (**4C**)
Azul Vega 2 (**48C**)



Many applications still single or lightly-threaded

Not feasible to rewrite many existing applications to work in parallel

Not feasible to rewrite many existing applications to work in parallel

➜ Automatic Parallelization

Thread-Level Speculation

Employs speculation

- No need to prove that parallelization is valid

Thread-Level Speculation

Employs speculation

- No need to prove that parallelization is valid
- Correctness dynamically ensured at runtime

Thread-Level Speculation

Employs speculation

- No need to prove that parallelization is valid
- Correctness dynamically ensured at runtime

➜ Uses some kind of in-memory transactions to buffer and validate tentative changes

JaSPEx-MLS

Our proposal: JaSPEx-MLS

- Software-based speculative parallelization framework for Java

JaSPEx-MLS

Our proposal: JaSPEx-MLS

- Software-based speculative parallelization framework for Java
- Java Bytecode rewriting

JaSPEx-MLS

Our proposal: JaSPEx-MLS

- Software-based speculative parallelization framework for Java
- Java Bytecode rewriting
- Automatic transactification

JaSPEx-MLS

Our proposal: JaSPEx-MLS

- Software-based speculative parallelization framework for Java
- Java Bytecode rewriting
- Automatic transactification
  - STM-like API

JaSPEx-MLS

Our proposal: JaSPEx-MLS

- Software-based speculative parallelization framework for Java
- Java Bytecode rewriting
- Automatic transactification
  - STM-like API
- Handling non-transactional operations

JaSPEx-MLS

Our proposal: JaSPEx-MLS

- Software-based speculative parallelization framework for Java
- Java Bytecode rewriting
- Automatic transactification
  - STM-like API
- Handling non-transactional operations
- Method-level speculation

JaSPEx-MLS

Our proposal: JaSPEx-MLS

- Software-based speculative parallelization framework for Java
- Java Bytecode rewriting
- Automatic transactification
  - STM-like API
- Handling non-transactional operations
- Method-level speculation
- Nested speculation

JaSPEx-MLS

Our proposal: JaSPEx-MLS

- Software-based speculative parallelization framework for Java
- Java Bytecode rewriting
- Automatic transactification
  - STM-like API
- Handling non-transactional operations
- Method-level speculation
- Nested speculation

...on top of the OpenJDK Java VM

JaSPEx-MLS

Our proposal: JaSPEx-MLS

- Software-based speculative parallelization framework for Java
- Java Bytecode rewriting
- Automatic transactification
  - STM-like API
- Handling non-transactional operations
- Method-level speculation
- Nested speculation

...on top of the OpenJDK Java VM + Continuation support

JaSPEx-MLS

Target applications

- Existing single-threaded JVM bytecode

JaSPEx-MLS

Target applications

- Existing single-threaded JVM bytecode
- Object-oriented / method-heavy

JaSPEx-MLS

Target applications

- Existing single-threaded JVM bytecode
- Object-oriented / method-heavy
  - Limited support for loops

JaSPEx-MLS

Target applications

- Existing single-threaded JVM bytecode
- Object-oriented / method-heavy
  - Limited support for loops

- Static analysis is hard or impossible

Why use OpenJDK?

- Modern open-source VM by Oracle and community contributors

Why use OpenJDK?

- Modern open-source VM by Oracle and community contributors
- Just-in-time compilation

Why use OpenJDK?

- Modern open-source VM by Oracle and community contributors

- Just-in-time compilation
- Adaptive optimization

Why use OpenJDK?

- Modern open-source VM by Oracle and community contributors

- Just-in-time compilation
- Adaptive optimization
- Advanced garbage collection

Why use OpenJDK?

- Modern open-source VM by Oracle and community contributors

- Just-in-time compilation
- Adaptive optimization
- Advanced garbage collection

- Java 6 support

Why use OpenJDK?

- Modern open-source VM by Oracle and community contributors

- Just-in-time compilation
- Adaptive optimization
- Advanced garbage collection

- Java 6 support
- Optimized concurrency primitives

Why use OpenJDK?

- Modern open-source VM by Oracle and community contributors

- Just-in-time compilation
- Adaptive optimization
- Advanced garbage collection

- Java 6 support
- Optimized concurrency primitives

➜ JaSPEx-MLS works on commonly available hardware

Why use OpenJDK?

- Modern open-source VM by Oracle and community contributors

- Just-in-time compilation
- Adaptive optimization
- Advanced garbage collection

- Java 6 support
- Optimized concurrency primitives

➜ JaSPEx-MLS works on commonly available hardware
➜ Same codebase used to run regular Java applications

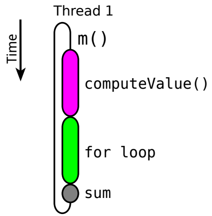
Method-Level Speculation

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

- Run method call in parallel with code following its return

Method-Level Speculation

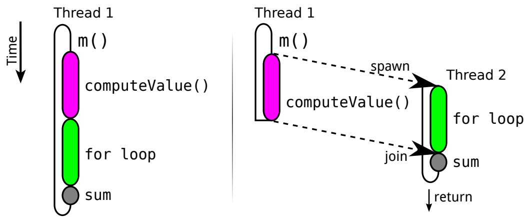
```
int example1() {
    int x = computeValue();
    int y = 0;
    for (...) y += ...;
    return x+y;
}
```



- Run method call in parallel with code following its return

Method-Level Speculation

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



- Run method call in parallel with code following its return

JaSPEx-MLS

Classloader

- Prepares code for speculative parallelization

JaSPEx-MLS

Classloader

- Prepares code for speculative parallelization

Runtime Orchestration Library

- Coordinates all runtime events
  - Creation, validation, commit, abort, ...

JaSPEx-MLS Classloader

Transactification

- Intercept access to slots and arrays

JaSPEx-MLS Classloader

Transactification

- Intercept access to slots and arrays

```
object.field = 100l;
```

JaSPEx-MLS Classloader

Transactification

- Intercept access to slots and arrays

```
object.field = 100l;
```

```
Transaction.storeLong(object, field, 100l);
```

JaSPEx-MLS Classloader

Transactification

- Intercept access to slots and arrays

```
object.field = 100l;
```

```
Transaction.storeLong(object, field, 100l);
```

- Static and type-specific

➜ Designed to be inlined by the VM

JaSPEx-MLS Classloader

Handling Non-Transactional Operations

JaSPEx-MLS Classloader

Handling Non-Transactional Operations

➜ Operations outside the control of the transactional system

JaSPEx-MLS Classloader

Handling Non-Transactional Operations

➜ Operations outside the control of the transactional system

- `native` methods

JaSPEx-MLS Classloader

Handling Non-Transactional Operations

➜ Operations outside the control of the transactional system

- `native` methods
- JDK code

JaSPEx-MLS Classloader

Handling Non-Transactional Operations

➜ Operations outside the control of the transactional system

- native methods
- JDK code

```
...

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

JaSPEx-MLS Classloader

Handling Non-Transactional Operations

➜ Operations outside the control of the transactional system

- native methods
- JDK code

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

JaSPEx-MLS Classloader

Modifications for MLS

- Replace method calls with call to `spawnSpeculation()`

JaSPEx-MLS Classloader

Modifications for MLS

- Replace method calls with call to `spawnSpeculation()`
- Replace method return value with `Future`

JaSPEx-MLS Classloader

Modifications for MLS

- Replace method calls with call to `spawnSpeculation()`
- Replace method return value with `Future`

```java
int mlsExample1() {
    return compute(0) + compute(1);
}
```

```java
int mlsExample1() {
    Future temp1 = spawnSpeculation(...);  // compute(0)
    Future temp2 = spawnSpeculation(...);  // compute(1)
    return temp1.get() + temp2.get();
}
```

JaSPEx-MLS Classloader

Modifications for MLS

- Replace method calls with call to `spawnSpeculation()`
- Replace method return value with `Future`

```
int mlsExample1() {
    return compute(0) + compute(1);
}
```

```
int mlsExample1() {
    Future temp1 = spawnSpeculation(...);  // compute(0)
    Future temp2 = spawnSpeculation(...);  // compute(1)
    return temp1.get() + temp2.get();
}
```

- Usually needs control flow analysis and code duplication

JaSPEx-MLS Classloader

STM support for Futures

- Allow Futures to be “written” to memory locations
- Delay obtaining value until it is needed (read or commit)

JaSPEx-MLS Classloader

STM support for Futures

```
// Original Method
void doCompute(Object[] results) {
  for (int i = 0; i < results.length; i++) {
    results[i] = compute(i);
  }}
```

JaSPEx-MLS Classloader

STM support for Futures

```
// Original Method
void doCompute(Object[] results) {
  for (int i = 0; i < results.length; i++) {
    results[i] = compute(i);
  }}

// *Attempted* parallelization
void doCompute(Object[] results) {
  for (int i = 0; i < results.length; i++) {
    Future f0 = spawnSpeculation(...); // compute(i)
    TM.storeObjectArray(results, i, f0.get());
  }}
```

JaSPEx-MLS Classloader

... let's try this again

```
// Original Method
void doCompute(Object[] results) {
  for (int i = 0; i < results.length; i++) {
    results[i] = compute(i);
  }}
```

JaSPEx-MLS Classloader

... let's try this again

```
// Original Method
void doCompute(Object[] results) {
  for (int i = 0; i < results.length; i++) {
    results[i] = compute(i);
  }}

// *Successful* parallelization
void doCompute(Object[] results) {
  for (int i = 0; i < results.length; i++) {
    Future f0 = spawnSpeculation(...); // compute(i)
    TM.storeFutureObjectArray(results, i, f0); // f0 handed
  }}                                           // to the STM
```

JaSPEx-MLS Runtime Orchestration

```
int mlsExample1() {
    Future temp1 = spawnSpeculation(...);  // compute(0)
    Future temp2 = spawnSpeculation(...);  // compute(1)
    return temp1.get() + temp2.get();
}
```

What happens inside spawnSpeculation?

- Check threadpool state

JaSPEx-MLS Runtime Orchestration

```
int mlsExample1() {
    Future temp1 = spawnSpeculation(...);  // compute(0)
    Future temp2 = spawnSpeculation(...);  // compute(1)
    return temp1.get() + temp2.get();
}
```

What happens inside spawnSpeculation?

- Check threadpool state
- Capture first-class continuation

JaSPEx-MLS Runtime Orchestration

```java
int mlsExample1() {
    Future temp1 = spawnSpeculation(...);  // compute(0)
    Future temp2 = spawnSpeculation(...);  // compute(1)
    return temp1.get() + temp2.get();
}
```

What happens inside spawnSpeculation?

- Check threadpool state
- Capture first-class continuation
- Submit child work task with continuation to pool

JaSPEx-MLS Runtime Orchestration

```
int mlsExample1() {
    Future temp1 = spawnSpeculation(...);  // compute(0)
    Future temp2 = spawnSpeculation(...);  // compute(1)
    return temp1.get() + temp2.get();
}
```

What happens inside spawnSpeculation?

- Check threadpool state
- Capture first-class continuation
- Submit child work task with continuation to pool
- Parent task:

JaSPEx-MLS Runtime Orchestration

```
int mlsExample1() {
    Future temp1 = spawnSpeculation(...);  // compute(0)
    Future temp2 = spawnSpeculation(...);  // compute(1)
    return temp1.get() + temp2.get();
}
```

What happens inside spawnSpeculation?

- Check threadpool state
- Capture first-class continuation
- Submit child work task with continuation to pool

- Parent task:
  - Clean current stack

JaSPEx-MLS Runtime Orchestration

```
int mlsExample1() {
    Future temp1 = spawnSpeculation(...);  // compute(0)
    Future temp2 = spawnSpeculation(...);  // compute(1)
    return temp1.get() + temp2.get();
}
```

What happens inside spawnSpeculation?

- Check threadpool state
- Capture first-class continuation
- Submit child work task with continuation to pool

- Parent task:
  - Clean current stack
  - Execute `compute(0)`

JaSPEx-MLS Runtime Orchestration

```
int mlsExample1() {
    Future temp1 = spawnSpeculation(...);  // compute(0)
    Future temp2 = spawnSpeculation(...);  // compute(1)
    return temp1.get() + temp2.get();
}
```

What happens inside spawnSpeculation?

- Check threadpool state
- Capture first-class continuation
- Submit child work task with continuation to pool

- Parent task:
  - Clean current stack
  - Execute `compute(0)`
  - Write result to child task

JaSPEx-MLS Runtime Orchestration

```
int mlsExample1() {
    Future temp1 = spawnSpeculation(...);  // compute(0)
    Future temp2 = spawnSpeculation(...);  // compute(1)
    return temp1.get() + temp2.get();
}
```

What happens inside spawnSpeculation?

- Check threadpool state
- Capture first-class continuation
- Submit child work task with continuation to pool

- Parent task:
  - Clean current stack
  - Execute `compute(0)`
  - Write result to child task
  - Return to poll

JaSPEx-MLS Runtime Orchestration

```
int mlsExample1() {
    Future temp1 = spawnSpeculation(...);  // compute(0)
    Future temp2 = spawnSpeculation(...);  // compute(1)
    return temp1.get() + temp2.get();
}
```

- Parent task:
  - Clean current stask
  - Execute `compute(0)`
  - Write result to child task
  - Return to poll

- Child task:

JaSPEx-MLS Runtime Orchestration

```
int mlsExample1() {
    Future temp1 = spawnSpeculation(...);  // compute(0)
    Future temp2 = spawnSpeculation(...);  // compute(1)
    return temp1.get() + temp2.get();
}
```

- Parent task:
  - Clean current stask
  - Execute `compute(0)`
  - Write result to child task
  - Return to poll

- Child task:
  - Start transaction

JaSPEx-MLS Runtime Orchestration

```
int mlsExample1() {
    Future temp1 = spawnSpeculation(...);  // compute(0)
    Future temp2 = spawnSpeculation(...);  // compute(1)
    return temp1.get() + temp2.get();
}
```

- Parent task:
  - Clean current stask
  - Execute `compute(0)`
  - Write result to child task
  - Return to poll

- Child task:
  - Start transaction
  - Resume continuation

JaSPEx-MLS Runtime Orchestration

```
int mlsExample1() {
    Future temp1 = spawnSpeculation(...);  // compute(0)
    Future temp2 = spawnSpeculation(...);  // compute(1)
    return temp1.get() + temp2.get();
}
```

- Parent task:
  - Clean current stask
  - Execute `compute(0)`
  - Write result to child task
  - Return to poll

- Child task:
  - Start transaction
  - Resume continuation
  - Return from spawnSpeculation

JaSPEx-MLS Runtime Orchestration

Committing a speculation:

- Task completed work

JaSPEx-MLS Runtime Orchestration

Committing a speculation:

- Task completed work
- Need to obtain value from a Future, and oldest-running task

JaSPEx-MLS Runtime Orchestration

Committing a speculation:

- Task completed work
- Need to obtain value from a Future, and oldest-running task
- Tried to execute non-transactional operations

JaSPEx-MLS Runtime Orchestration

Committing a speculation:

- Task completed work
- Need to obtain value from a Future, and oldest-running task
- Tried to execute non-transactional operations

- Only oldest-running task is allowed to commit

JaSPEx-MLS Runtime Orchestration

Committing a speculation:

- Task completed work
- Need to obtain value from a Future, and oldest-running task
- Tried to execute non-transactional operations

- Only oldest-running task is allowed to commit

- Wait for result from parent

JaSPEx-MLS Runtime Orchestration

Committing a speculation:

- Task completed work
- Need to obtain value from a Future, and oldest-running task
- Tried to execute non-transactional operations

- Only oldest-running task is allowed to commit

- Wait for result from parent
- If exception ➜ abort

JaSPEx-MLS Runtime Orchestration

Committing a speculation:

- Task completed work
- Need to obtain value from a Future, and oldest-running task
- Tried to execute non-transactional operations

- Only oldest-running task is allowed to commit

- Wait for result from parent
- If exception ➜ abort
- Else try to commit

Relaxed STM Model

Distinguish between

- Program-order mode: Reads and writes directly to memory

Relaxed STM Model

Distinguish between

- Program-order mode: Reads and writes directly to memory
  - Not running speculatively — only one thread allowed in this mode

Relaxed STM Model

Distinguish between

- Program-order mode: Reads and writes directly to memory
  - Not running speculatively — only one thread allowed in this mode

- Speculation mode: Read directly from memory (if not in write-set), writes go to write-set

Relaxed STM Model

Distinguish between

- Program-order mode: Reads and writes directly to memory
  - Not running speculatively — only one thread allowed in this mode

- Speculation mode: Read directly from memory (if not in write-set), writes go to write-set
  - Inconsistent reads

Relaxed STM Model

Distinguish between

- Program-order mode: Reads and writes directly to memory
  - Not running speculatively — only one thread allowed in this mode
- Speculation mode: Read directly from memory (if not in write-set), writes go to write-set
  - Inconsistent reads

Transaction commit is simpler, because only one transaction may commit at a time

Relaxed STM Model

Distinguish between

- Program-order mode: Reads and writes directly to memory
  - Not running speculatively — only one thread allowed in this mode
- Speculation mode: Read directly from memory (if not in write-set), writes go to write-set
  - Inconsistent reads

Transaction commit is simpler, because only one transaction may commit at a time

➜ No synchronization needed

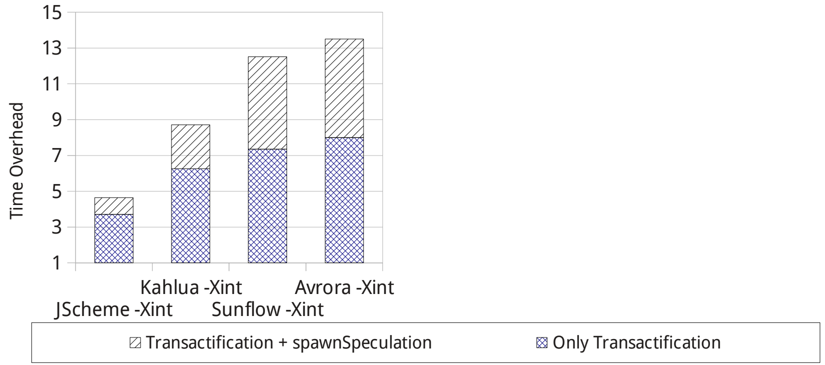
Benchmarks

Benchmarks

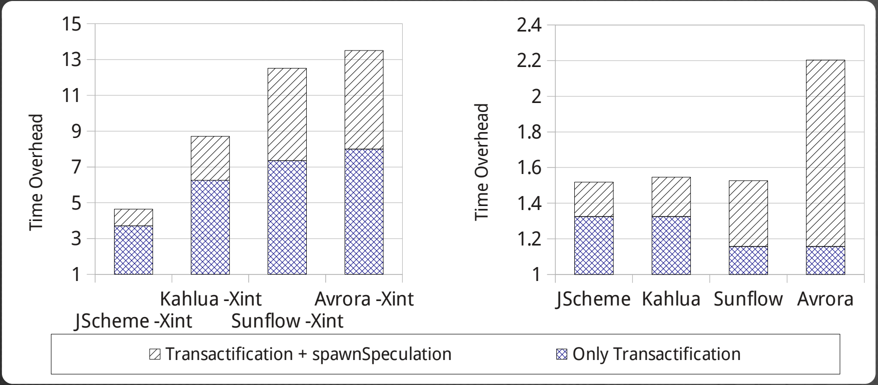
- Test overheads from bytecode changes
- Early results from full system

- Test system: Intel Core i5 750, 8GB RAM, Ubuntu Linux 12.04 64-bit

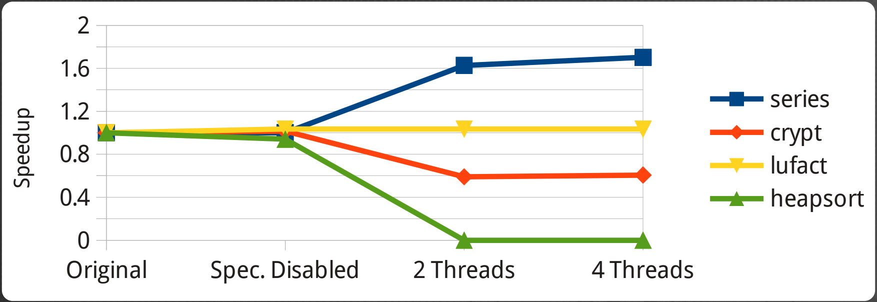
Benchmarks — Overhead with Interpreter



Benchmarks — Overhead with JIT



Benchmarks — Parallelization



- Benchmarks from Java Grande Framework
- Source was **not** modified

Conclusions

- JaSPEx-MLS: Automatic parallelization framework for Java

Conclusions

- JaSPEx-MLS: Automatic parallelization framework for Java
- Using custom lightweight STM and MLS

Conclusions

- JaSPEx-MLS: Automatic parallelization framework for Java
- Using custom lightweight STM and MLS
- Supports nested speculation

Conclusions

- JaSPEx-MLS: Automatic parallelization framework for Java
- Using custom lightweight STM and MLS
- Supports nested speculation

- Needs no special hardware support

Conclusions

- JaSPEx-MLS: Automatic parallelization framework for Java
- Using custom lightweight STM and MLS
- Supports nested speculation

- Needs no special hardware support
- On top of OpenJDK VM

Conclusions

- Integrating Futures with the STM unlocks more parallelism

Conclusions

- Integrating Futures with the STM unlocks more parallelism
- Optimizing VM makes ***all*** the difference

Conclusions

- Integrating Futures with the STM unlocks more parallelism
- Optimizing VM makes ***all*** the difference
  - Able to hide much of the added overhead

Conclusions

- Integrating Futures with the STM unlocks more parallelism
- Optimizing VM makes ***all*** the difference
  - Able to hide much of the added overhead
- Already able to extract parallelism in some benchmarks

Future Work

- Lift some limitations imposed by non-transactional ops

Future Work

- Lift some limitations imposed by non-transactional ops
- Add support for profiling pass, and to save statistic information between runs

Future Work

- Lift some limitations imposed by non-transactional ops
- Add support for profiling pass, and to save statistic information between runs
- Integrate task scheduler

Future Work

- Lift some limitations imposed by non-transactional ops
- Add support for profiling pass, and to save statistic information between runs
- Integrate task scheduler
- Re-use waiting threads by returning them to the thread-pool

Thank you!