Johannes Kepler University Linz

Lukas Stadler

Johannes Kepler University Linz, Austria

Johannes Kepler University Linz

Dynamic Code Evolution

Coroutines for the JVM

Dynamic Code Evolution

JKU
JOHANNES KEPLER
UNIVERSITY LINZ

Thomas Würthinger
Johannes Kepler University Linz, Austria

ORACLE®

Dynamic Code Evolution - Past

- Modification of the Java HotSpot™ VM.

- Allows **arbitrary** class redefinition changes:
  - Add/remove methods.
  - Add/remove fields.
  - Change the implemented interfaces or the super class.

- No performance penalty on normal program execution.

Dynamic Code Evolution - Improvements

Stability

- Passes Oracle's internal class redefinition test suite!
- Larger test suite for advanced changes.

Functionality

- Introduction of transformer methods.
- Can call old deleted method and access old deleted static field.

Performance

- Garbage collection run now only necessary if object instances are affected.

Dynamic Code Evolution - Future

Website

- http://ssw.jku.at/dcevm
- Binaries for Linux-32bit, MacOS-32bit, Win-32bit, Win-64bit
- Code available in the MLVM repository

Sponsors

ORACLE

Guidewire
Deliver insurance your way

Integration

- Looking for a sponsoring engineer at Oracle!

JVM Coroutines

part of the MLVM project

Lukas Stadler

Johannes Kepler University Linz, Austria

Coroutines

- Old (ancient?) concept
- Extensions of subroutines
    - multiple entry/exit points
- Lightweight threads
- Many variations
    - generators, coexpressions, fibers, iterators, green threads, greenlets, tasklets, …
    - sometimes as generic as coroutines

Variations of coroutines

- Stackless / stackful
  - yield possible outside of main method?
- First-class coroutines
  - not tied to a specific language construct
- (a)symmetric
  - asymmetric: can only return control to their caller
  - symmetric: arbitrary control passing

Coroutines - why?

- Natural control abstraction for some problems
  - scanner/parser (Conway 1963)
  - non-parallel problems
  - do not expose parallelism where there is none!
- Inversion of algorithms
  - converting callback-based algorithms (e.g. SAX parser) into iterative algorithms
  - can be done manually, but coroutines do it for free!

Coroutines - why?

- Language implementations need to emulate coroutines
  - using threads
    - synchronize multiple threads so they look like coroutines
  - compile-time transformations
    - complex compilers
    - smaller chunks: less optimization opportunities for JIT
    - local variables on the heap (less register use, …)
  - that's bad, right?

Goals

- Features:
  - stackful
  - first-class
  - symmetric and asymmetric coroutines
    - it has been shown that these are equivalent
    - but: need both to be useful out of the box
- Performance:
  - fast switching
  - many coroutines
  - implementation complexity (not!)

trade-off

JVM coroutine implementation

- Allocate additional stacks
  - manages multiple stacks within threads
  - migration between threads is not supported (yet?)
- Stacks take lots of space
  - address space: 32-100 kb minimum
  - memory: 16-32 kb minimum
  - worst-case theoretical maximum: 20.000 stacks (32 bit Solaris)
  - not enough!

JVM coroutine implementation

- Suspended coroutines:
  small amount of stack is actually in use
- Sharing stacks if there are too many coroutines
  - copying stack contents to/from the stack
  - bad for performance, but allows many coroutines
  - 1.000.000 coroutines on a 32 bit machine
- Stacks need to be in the same place every time they are executed (native frames)

API

- symmetric coroutines: `Coroutine`

```
public class Coroutine {
    public Coroutine();
    public Coroutine(Runnable target);
    public Coroutine(long stacksize);
    public Coroutine(Runnable target, long stacksize);

    public static void yield();
    public static void yieldTo(Coroutine target);

    protected void run();
}
```

API

- symmetric coroutines: `Coroutine`

```
public class CoroutineTest extends Coroutine {
    @Override
    public void run() {
        System.out.println("Coroutine running 1");
        yield();
        System.out.println("Coroutine running 2");
    }

    public static void main(String[] args) {
        new CoroutineTest();
        System.out.println("start");
        yield();
        System.out.println("middle");
        yield();
        System.out.println("end");
    }
}
```

```
start
Coroutine running 1
middle
Coroutine running 2
end
```

API

- **asymmetric coroutines:** `AsymCoroutine`

```
public abstract class AsymCoroutine<InT, OutT> implements Iterable<OutT>
{
   public AsymCoroutine();
   public AsymCoroutine(long stacksize);

   public InT ret(OutT value);
   public InT ret();
   public OutT call(InT input);
   public OutT call();

   protected abstract OutT run(InT value);

   @Override
   public Iterator<OutT> iterator();
}
```

null as input parameter

API

- **asymmetric coroutines:** `AsymCoroutine`

```
public class CoSAXParser extends AsymCoroutine<Void, String> {
    @Override
    public String run(Void value) {
        SAXParserFactory.newInstance().newSAXParser().parse(..., new DefaultHandler() {
            public void startElement(String uri, String localName, String name, Attributes att) {
                ret(name);
            }
        });
        return null;
    }

    public static void main(String[] args) {
        CoSAXParser parser = new CoSAXParser();

        String element;
        do {
            element = parser.call();
            System.out.println(element);
        } while(element != null);
    }
}
```

```
office:document-content
office:scripts
office:font-face-decls
style:font-face
...
...
```

API

- **asymmetric coroutines:** `AsymCoroutine`

```
public class CoSAXParser extends AsymCoroutine<Void, String> {
    @Override
    public String run(Void value) {
        SAXParserFactory.newInstance().newSAXParser().parse(..., new DefaultHandler() {
            public void startElement(String uri, String localName, String name, Attributes att) {
                ret(name);
            }
        });
        return null;
    }

    public static void main(String[] args) {
        CoSAXParser parser = new CoSAXParser();

        String element;
        do {
            element = parser.call();
            System.out.println(element);
        } while(element != null);
    }
}
```

```
office:document-content
office:scripts
office:font-face-decls
style:font-face
...
...
```

API

- asymmetric coroutines: `AsymCoroutine`

```
public class CoSAXParser extends AsymCoroutine<Void, String> {
    @Override
    public String run(Void value) {
        SAXParserFactory.newInstance().newSAXParser().parse(..., new DefaultHandler() {
            public void startElement(String uri, String localName, String name, Attributes att) {
                ret(name);
            }
        });
        return null;
    }

    public static void main(String[] args) {
        CoSAXParser parser = new CoSAXParser();

        for (String element : parser)
            System.out.println(element);
    }
}
```

```
office:document-content
office:scripts
office:font-face-decls
style:font-face
...
...
```

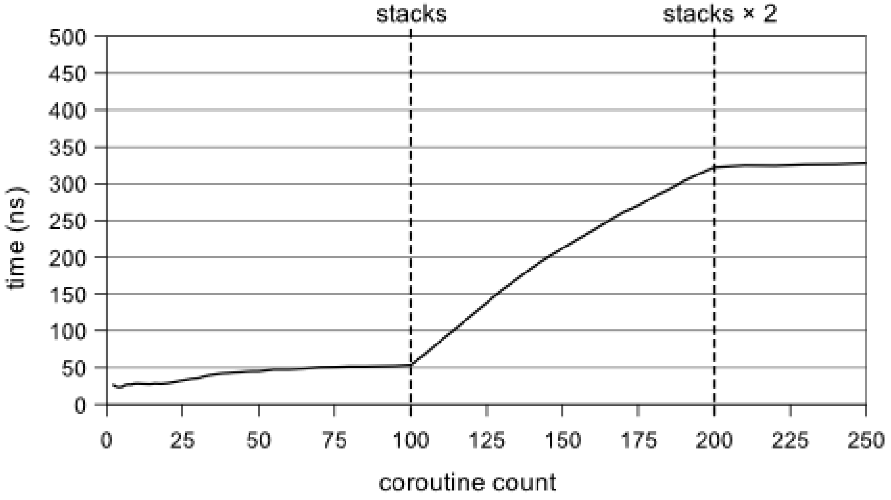
API

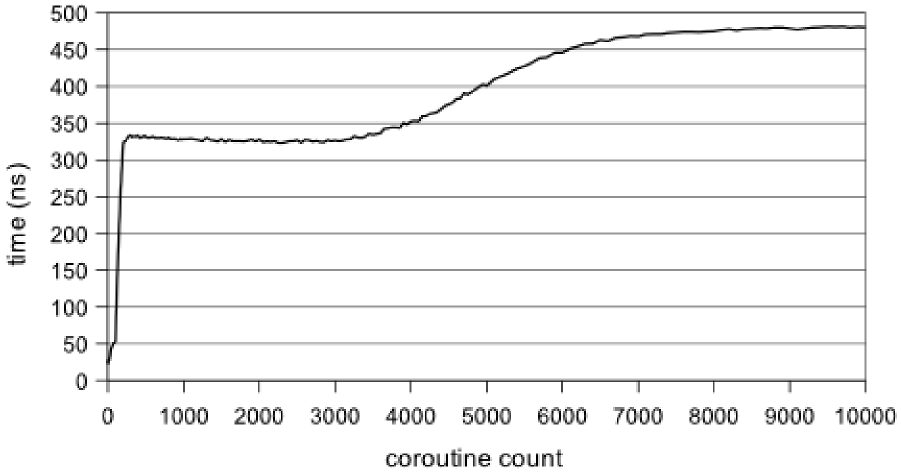
- Coroutine locals: `CoroutineLocal`
- … similar to `ThreadLocal`
- What to do with coroutines when a thread ends?
  - *“no coroutine left behind”*
  - `AsymCoroutines` will receive CoroutineDeath as soon as there are no more (symmetric) `Coroutines`

Performance

- Memory:
  - stacks + storage for rescued coroutines
- Run time: (Intel i5 750 CPU)
  - create: 1.5 µs / 0.3 µs (Thread: 2.5 µs)
  - start: 3 µs (Thread: 60 µs)
  - switch: 20 ns (best case)
    - difficult to get reliable thread switch time

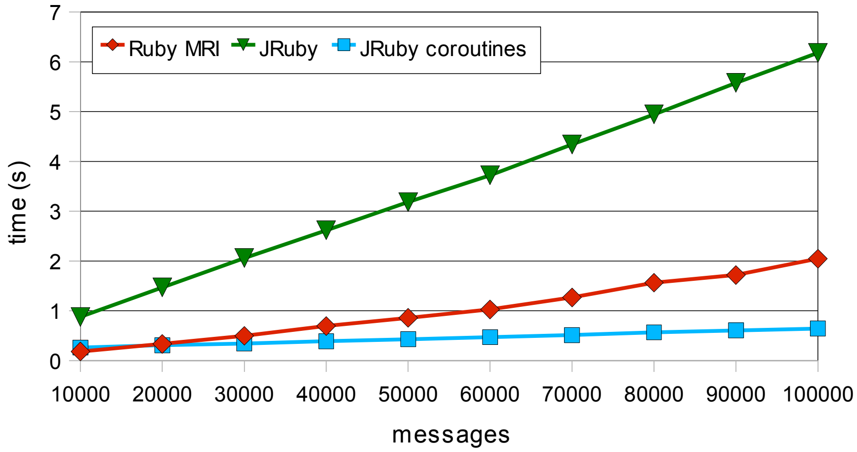
Performance



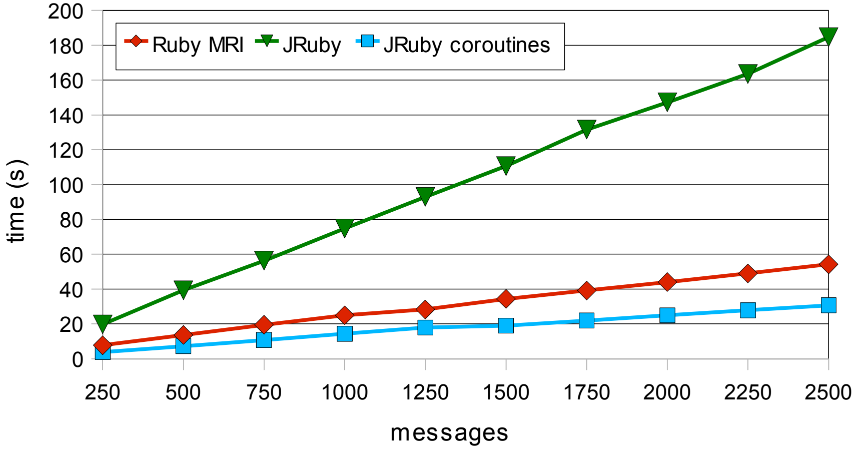


- Time per switch, depending on coroutine count
  - working set outgrows L1 and L2 caches
  - 500 ns even for large numbers of coroutines
- 100 coroutine stacks per thread

Performance: JRuby



5 fiber chain



5000 fiber chain

- Small contrived benchmark
- Passes messages through a chain of fibers
- Improvement from 3 x slower to 2 x faster!
  - won't convert to real-world applications

Weak spots

- Not quite stable
  - reliable `StackOverflowError`
  - occasional GC errors
- Rescuing stack frames with locks
- Large scale tests

Future, Ideas, ...

- Paper at PPPJ 2010 in Vienna:
  Efficient Coroutines for the Java Platform
  - I'll be happy to provide a preprint copy
- Seralizable coroutines?
  - some of the continuation use cases
  - ... along with many of the problems
- Code available in the MLVM repository

Thank you.
Questions?

For details see:
Efficient Coroutines for the Java Platform
Conference on Principles and Practice of Programming in Java 2010

Lukas Stadler
Johannes Kepler University Linz, Austria

Thomas Würthinger
Johannes Kepler University Linz, Austria

Christian Wimmer
University of California, Irvine