Implementing Ruby Using Truffle and Graal

Chris Seaton @ChrisGSeaton

ECOOP Summer Schools 2014
Safe Harbor Statement

The following is intended to provide some insight into a line of research in Oracle Labs. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. Oracle reserves the right to alter its development plans and practices at any time, and the development, release, and timing of any features or functionality described in connection with any Oracle product or service remains at the sole discretion of Oracle. Any views expressed in this presentation are my own and do not necessarily reflect the views of Oracle.
We’re going to talk about

1. Motivation
2. Truffle and Graal Theory
3. Truffle and Graal in Practice
4. Applying it to Ruby
Motivation
JavaScript: **One language to rule them all** | VentureBeat
venturebeat.com/2011/.../javascript-one-language-to-rule-them-all...
by Peter Yared - in 23 Google+ circles
Jul 29, 2011 - Why code in two different scripting languages, one on the client and one on the server? It's time for **one language to rule them all**. Peter Yared ...

[PDF] **Python: One Script (Language) to rule them all** - Ian Darwin
www.darwinsys.com/python/python4unix.pdf
Another **Language**? • Python was invented in 1991 by Guido van. Rossum. • Named after the comedy troupe, not the snake. • Simple. • They all say that!

**Q & Stuff: One Language to Rule Them All - Java**
qstuff.blogspot.com/2005/10/one-language-to-rule-them-all-java.html
Oct 10, 2005 - **One Language to Rule Them All - Java**. For a long time I'd been hoping to add a scripting language to LibQ, to use in any of my (or other ...

**Dart : one language to rule them all** - MixIT 2013 - Slideshare
fr.slideshare.net/sdeleuze/dart-mixit2013en
DartSébastien Deleuze - @sdeleuzeMix-IT 2013One **language to rule them all** ...
Why can't there be an “ultimate” programming language?

closed as not constructive by Tim, Bo Persson, Devon_C_Miller, Mark, Graviton Jan 17 at 5:58
Computer Language Benchmarks Game
Goal:

Computer Language Benchmarks Game
Prototype a new language
Parser and language work to build syntax tree (AST), AST Interpreter
Write a “real” VM
In C/C++, still using AST interpreter, spend a lot of time implementing runtime system, GC, ...
People start using it
People complain about performance
Define a bytecode format and write bytecode interpreter
Performance is still bad
Write a JIT compiler
Improve the garbage collector

Prototype a new language in Java
Parser and language work to build syntax tree (AST)
Execute using AST interpreter
People start using it
And it is already fast
Truffle and Graal Theory
Guest Language

Bytecode

JVM
Guest Language

Java IR, machine code cache, invalidation and deoptimisation, optimisation phases, replacements, etc... etc...
Node Rewriting for Profiling Feedback

Node Transitions

Uninitialized Nodes

Uninitialized

Integer

Double

Generic

String

AST Interpreter Rewritten Nodes

Compilation using Partial Evaluation

Compiled Code

Frequently executed call
Deoptimization to AST Interpreter

Node Rewriting to Update Profiling Feedback

Recompilation using Partial Evaluation

Static Analysis

Reachable methods, fields, and classes

Ahead-of-Time Compilation

All Java classes from application, JDK, and Substrate VM

Application running without compilation or class loading
One VM

• Good interpreted performance on a standard JVM
• Extremely good dynamically compiled performance on Graal
• High level representation of languages
• Substrate VM for startup performance, low footprint and easy distribution

• JavaScript, Ruby, R, J, C, Python, SmallTalk
Truffle and Graal in Practice
Simple Language (SL)

• Minimal language for demonstration and documentation
• Similar to JavaScript
• Included in the OpenJDK Graal repository
Setup

- `hg clone http://hg.openjdk.java.net/graal/graal`
- `cd graal`
- `./mx.sh --vm server build`
- `./mx.sh ideinit`

- Or just Google ‘graal openjdk’
public class SLIfNode extends SLStatementNode {
  @Child private SLExpressionNode conditionNode;
  @Child private SLStatementNode thenPartNode;
  @Child private SLStatementNode elsePartNode;

  public SLIfNode(SLExpressionNode conditionNode, SLStatementNode thenPartNode, SLStatementNode elsePartNode) {
    this.conditionNode = conditionNode;
    this.thenPartNode = thenPartNode;
    this.elsePartNode = elsePartNode;
  }

  public void executeVoid(VirtualFrame frame) {
    if (conditionNode.executeBoolean(frame)) {
      thenPartNode.executeVoid(frame);
    } else {
      elsePartNode.executeVoid(frame);
    }
  }
}
public class SLBlockNode extends SLStatementNode {
    @Children private final SLStatementNode[] bodyNodes;

    public SLBlockNode(SLStatementNode[] bodyNodes) {
        this.bodyNodes = adoptChildren(bodyNodes);
    }

    @ExplodeLoop
    public void executeVoid(VirtualFrame frame) {
        for (SLStatementNode statement : bodyNodes) {
            statement.executeVoid(frame);
        }
    }
}
```java
public class SLReturnNode extends SLStatementNode {
    @Child private SLExpressionNode valueNode;
    ...
    public void executeVoid(VirtualFrame frame) {
        throw new SLReturnExceptn(valueNode.executeGeneric(frame));
    }
}
```

```java
public class SLFunctionBodyNode extends SLExpressionNode {
    @Child private SLStatementNode bodyNode;
    ...
    public Object executeGeneric(VirtualFrame frame) {
        try {
            bodyNode.executeVoid(frame);
        } catch (SLReturnException ex) {
            return ex.getResult();
        }
        return SLNull.SINGLETON;
    }
}
```

```java
public class SLReturnException extends ControlFlowException {
    private final Object result;
}
```
public class SLAddNode extends SLExpressionNode {
  @Child private SLExpressionNode leftNode;
  @Child private SLExpressionNode rightNode;

  @Override
  public Object executeGeneric(VirtualFrame frame) {
    Object left = leftNode.executeGeneric(frame);
    Object right = rightNode.executeGeneric(frame);

    if (left instanceof Long && right instanceof Long) {
      try {
        return ExactMath.addExact((Long) left, (Long) right);
      } catch (ArithmeticException ex) {
      }
    }

    if (left instanceof Long) {
      left = BigInteger.valueOf((Long) left);
    }
    if (right instanceof Long) {
      right = BigInteger.valueOf((Long) right);
    }

    if (left instanceof BigInteger && right instanceof BigInteger) {
      return ((BigInteger) left).add((BigInteger) right);
    }

    if (left instanceof String || right instanceof String) {
      return left.toString() + right.toString();
    }

    throw new UnsupportedSpecializationException(this, ...);
  }
}
public Object executeGeneric(VirtualFrame frame) {
    Object left = leftNode.executeGeneric(frame);
    Object right = rightNode.executeGeneric(frame);

    if (left instanceof Long && right instanceof Long) {
        try {
            return ExactMath.addExact((Long) left, (Long) right);
        } catch (ArithmeticException ex) { }
    }

    if (left instanceof Long) {
        left = BigInteger.valueOf((Long) left);
    }
    if (right instanceof Long) {
        right = BigInteger.valueOf((Long) right);
    }
    if (left instanceof BigInteger && right instanceof BigInteger) {
        return ((BigInteger) left).add((BigInteger) right);
    }
    if (left instanceof String || right instanceof String) {
        return left.toString() + right.toString();
    }
...

```java
@Specialization(rewriteOn = ArithmeticException.class)
protected long add(long left, long right) {
    return ExactMath.addExact(left, right);
}

@Specialization
protected BigInteger add(BigInteger left, BigInteger right) {
    return left.add(right);
}

@Specialization(guards = "isString")
protected String add(Object left, Object right) {
    return left.toString() + right.toString();
}

protected boolean isString(Object a, Object b) {
    return a instanceof String || b instanceof String;
}
```
Ruby in Truffle and Graal
Introduction to Ruby

• Imperative, object oriented, dynamically typed
• Inspirations from Smalltalk and Perl
• Widely used with the Rails framework for web applications
• But also used in graphics, bioinformatics, systems, etc
Ruby Implementations - MRI

•Implemented in C
•Bytecode interpreter
•Very simple optimisations – inline caches in instructions
•Probably the slowest commonly used interpreter there is
Ruby Implementations - Rubinius

• Implemented in C++ and Ruby
• Uses an LLVM-based JIT
Ruby Implementations - Topaz

- Implemented in RPython
- Interpreter is statically compiled to native code via C
- Ruby code is compiled using a tracing JIT compiler
Ruby Implementations - JRuby

• Implemented in Java
• Driver and primary user of JSR 292 (invokedynamic) until Nashorn
• AST interpreter -> bytecode compiler - > JIT by JVM
• Now looking at their own IR before bytecode
Ruby Implementations – JRuby+Truffle

- Uses JRuby’s parser and limited parts of their runtime
- Currently not much more than a tenant within JRuby
- AST interpreter, written using Truffle
- Works on a normal JVM
- Can implicitly use Graal VM
Benchmarks

• chunky_png and psd.rb
• Real code, unmodified from the original libraries
• Range of styles of Ruby code:
  – High performance tight numerical loops with local variables
  – Object oriented code such as method calls and instance variables
  – Ruby dynamic programming features such as #send
Performance on chunky_png and psd.rb

chrisseaton.com/rubytruffle/pushing-pixels
new_r = blend_channel(r(bg), r(fg), mix_alpha)

... 

def method_missing(method, *args, &block)
    return ChunkyPNG::Color.send(method, *args) if ChunkyPNG::Color.respond_to?(method)
    normal(*args)
end
module Foo
  extend self
  def foo(a, b, c)
    hash = {a: a, b: b, c: c}
    array = hash.map { |k, v| v }
    x = array[0]
    y = [a, b, c].sort[1]
    x + y
  end
end

class Bar
  def method_missing(method, *args)
    if Foo.respond_to?(method)
      Foo.send(method, *args)
    else
      0
    end
  end
end

Bar.new.foo(14, 8, 6) => 22
Hello World on Substrate VM

### Execution Time

![Execution Time Chart]

- MRI: 13 msec
- JRuby: 353 msec
- Truffle on JVM: 688 msec
- Truffle on SVM: 14 msec

### Memory Footprint

![Memory Footprint Chart]

- MRI: 5 MByte
- JRuby: 35 MByte
- Truffle on JVM: 53 MByte
- Truffle on SVM: 9 MByte
RubySpec

rubyspec.org – thanks to Brian Shirai et al
<table>
<thead>
<tr>
<th>Language Feature</th>
<th>Implemented</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixnum to Bignum promotion</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Support for floating point</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Closures</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Bindings and eval</td>
<td>✓</td>
<td>Works from aliased methods</td>
</tr>
<tr>
<td>callcc and Continuation</td>
<td>✓</td>
<td>Limited support, the same as JRuby</td>
</tr>
<tr>
<td>Fibers</td>
<td>✓</td>
<td>Limited support, the same as JRuby</td>
</tr>
<tr>
<td>Frame local variables</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>C extensions</td>
<td>✓</td>
<td>Early work, but runs real C extensions</td>
</tr>
<tr>
<td>Ruby 1.9 encoding</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Garbage collection</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Concurrency and parallelism</td>
<td>✓</td>
<td>We currently use a GIL</td>
</tr>
<tr>
<td>Tracing and debugging</td>
<td>✓</td>
<td>Always enabled</td>
</tr>
<tr>
<td>ObjectSpace</td>
<td>✓</td>
<td>Always enabled</td>
</tr>
<tr>
<td>Method invalidation</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Constant invalidation</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Ruby on Rails</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Language Feature</th>
<th>Implemented</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixnum to Bignum promotion</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Closures</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Bindings and eval</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Garbage collection</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Tracing and debugging</td>
<td>✓</td>
<td>Always enabled</td>
</tr>
<tr>
<td>ObjectSpace</td>
<td>✓</td>
<td>Always enabled</td>
</tr>
<tr>
<td>Method invalidation</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Constant invalidation</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

Setup

- git clone https://github.com/jruby/jruby.git
- cd jruby
- mvn package

- Or just Google ‘jruby truffle wiki’
Fixnum to Bignum Promotion

• Fixnum – fixed integer: C `int64_t` or Java `long`
• Bignum – arbitrary integer: C `mpz_t` or Java `BigInteger`

• Fixnum overflows to Bignum
• Bignum underflows (?) to Fixnum
• Entirely different classes – programmer can tell the difference

• Unlike JavaScript and Python
Closures

• Anonymous functions that capture a lexical scope
• Called ‘blocks’ in Ruby – higher order methods

```ruby
x = 14
my_array = [1, 2, 3, 4]
my_array.each do |n|
  puts x + n
end
```
Closures

• Anonymous functions that capture a lexical scope
• Called ‘blocks’ in Ruby – higher order methods

```javascript
x = 14;
my_array = [1, 2, 3, 4];
my_array.each(function(n) {
    console.log(x + n);
});
```
Bindings and Eval

• Binding: get an environment as an object
• eval: as you’d expect, also lets you supply a Binding

```ruby
def foo
  a = 1
  b = 2
  binding
end
puts foo.local_variable_get(:a)
```
Bindings and Eval

• Binding: get an environment as an object
• eval: as you’d expect, also lets you supply a Binding

```ruby
alias :secret_binding :binding

def foo
  a = 1
  b = 2
  secret_binding
end

puts foo.local_variable_get(:a)
```
Method and Constant Invalidation

• Ruby lets you define methods – ‘monkey patching’

```ruby
class Fixnum
  def *(b)
    self + b
  end
end

puts 14 * 2 #=> 16 (not 28)
```
class Fixnum
  def *(b)
    eval "class Object::Fixnum
      def /(b)
        self - b
      end
    end
    "
    self + b
  end
end

puts 14 * 2 / 4 #=> 12 (not 4 or 7)
set_trace_func proc { |line, binding|
  puts “We’re at line number #{line}”
}

x = 1  => “We’re at line number 6”
y = 2  => “We’re at line number 7”
Inactive assumption check completely elided in compiled code

Debug action installed by user

Compile: produces partially evaluated machine code from specialized AST.

Deoptimize: transfers control from the machine code back to the AST interpreter.

Replace: the inactive node with an active node to install the debug action.

Compile: produces new machine code from the modified AST and the installed debug action.
Enabling set_trace_func

Using set_trace_func

Enabling debugger

Breakpoint

Constant conditional

Simple conditional

MRI
Rubinius
JRuby
Topaz
JRuby+Truffle

Self Relative Time (s/s)

10000

1000

100

10

1

n/a

n/a

n/a

n/a

n/a

n/a

n/a

n/a

n/a

n/a

n/a

n/a
ObjectSpace

- ObjectSpace allows you to enumerate all live objects
- Behind a flag –X+O in JRuby
- How to find all live objects in a JVM?

```
ObjectSpace#each_object do |o|
  puts o
end
```
Wrap Up
Get Involved

• Now is a great time to get involved in Truffle and Graal
• Personal opinion: I’d like to see them in JDK 9 in about 2 years

• Areas open for research: concurrency, parallelism, heterogeneous offload, language interoperability
• Build your language research on top of Truffle and Graal
• Implement a language: Haskell, Erlang, Swift, Clojure, PHP
• Design and implement an entirely new language
Get Involved

• http://openjdk.java.net/projects/graal/
• graal-dev@openjdk.java.net

• Documentation admittedly a little bit limited so far
• Look at SL and Ruby
• chris.seaton@oracle.com
• @ChrisGSeaton
Many people behind Truffle and Graal

**Oracle Labs**
- Danilo Ansaloni
- Daniele Bonetta
- Laurent Daynès
- Erik Eckstein
- Michael Haupt
- Mick Jordan
- Peter Kessler
- Christos Kotselidis
- David Leibs
- Tom Rodriguez
- Roland Schatz
- Doug Simon
- Lukas Stadler
- Michael Van De Vanter
- Christian Wirth
- Mario Wolczko
- Thomas Würthinger
- Laura Hill (Manager)

**Interns**
- Shams Imam
- Stephen Kell

**JU Linz**
- Prof. Hanspeter Mössenböck
- Gilles Duboscq
- Matthias Grimmer
- Christian Häubl
- Josef Haider
- Christian Huber
- Manuel Rigger
- Bernhard Urban
- Andreas Wöß

**University of Manchester**
- Chris Seaton

**University of Edinburgh**
- Christophe Dubach
- Juan José
- Fumero Alfonso
- Ranjeet Singh
- Toomas Remmelg

**LaBRI**
- Floréal Morandat

**University of California, Irvine**
- Prof. Michael Franz
- Codrut Stancu
- Gulfem Savrun Yeniceri
- Wei Zhang

**Purdue University**
- Prof. Jan Vitek
- Tomas Kalibera
- Petr Maj
- Lei Zhao

**T. U. Dortmund**
- Prof. Peter Marwedel
- Helena Koshaus
- Ingo Korb

**University of California, Davis**
- Prof. Duncan Temple Lang
- Nicholas Ulle
Safe Harbor Statement

The preceding is intended to provide some insight into a line of research in Oracle Labs. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. Oracle reserves the right to alter its development plans and practices at any time, and the development, release, and timing of any features or functionality described in connection with any Oracle product or service remains at the sole discretion of Oracle. Any views expressed in this presentation are my own and do not necessarily reflect the views of Oracle.
Hardware and Software
Engineered to Work Together