Turning the JVM into a Polyglot VM with Graal

Chris Seaton
Research Manager
Oracle Labs
April 2017
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.
Programming languages
Stack Overflow is a question and answer site for professional and enthusiast programmers. It's 100% free, no registration required.

Why can't there be an “ultimate” programming language?
Stack Overflow is a question and answer site for professional and enthusiast programmers. It's 100% free, no registration required.

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
JavaScript: One language to rule them all | VentureBeat
venturebeat.com/2011/.../javascript-one-language-to-rule-them-...
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 ...
Computer Language Benchmarks Game

The diagram compares the performance of different computer languages in a benchmark test. The languages include C, C++, Java, JavaScript (JS), PHP, Python, and R. The y-axis represents the performance in mean values, ranging from 1 to 1000.
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
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
The GraalVM concept
How we do polyglot in GraalVM
Truffle::Interop.eval('application/language', source)

value = Truffle::Interop.import(name)

Truffle::Interop.export(name)
```javascript
Interop.eval('application/language', source)

value = Interop.import(name)

Interop.export(name)
```
puts Truffle::Interop.eval('application/javascript', '14 + 2')
# 16
Ruby:

```ruby
puts Truffle::Interop.eval('application/javascript', '14 + 2')
# 16
```

JavaScript:

```javascript
```

*Oracle*
Truffle::Interop.eval('application/javascript', "
  function add(a, b) {
    return a + b;
  }

  Interop.export('add', add.bind(this));
")

add = Truffle::Interop.import('add')

puts add.call(14, 2)
# 16
Ruby

Truffle::Interop.eval('application/javascript', "
  function add(a, b) {
    return a + b;
  }

  Interop.export('add', add.bind(this));
"

add = Truffle::Interop.import('add')

puts add.call(14, 2)
# 16
function add(a, b) {
    return a + b;
}

puts add(14, 2)
# 16
JavaScript

```javascript
function add(a, b) {
  return a + b;
}
```

Ruby

```ruby
puts add(14, 2)
# 16
```
function Point(x, y) {
    this.x = x;
    this.y = y;
}

function random_points(n) {
    points = [];
    for (i = 0; i < n; i++) {
        points[i] = new Point(Math.random(), Math.random())
    }
    return points;
}

points = random_points(100)

point = points[0]
puts point.x, point.y
# 0.642460680339328
# 0.116305386298814
```javascript
function Point(x, y) {
    this.x = x;
    this.y = y;
}

function random_points(n) {
    points = [];
    for (i = 0; i < n; i++) {
        points[i] = new Point(Math.random(), Math.random())
    }
    return points;
}

points = random_points(100)

point = points[0]
puts point.x, point.y
# 0.642460680339328
# 0.116305386298814
```
Performance
def clamp(num, min, max)
    [min, num, max].sort[1]
end

def cmyk_to_rgb(c, m, y, k)
    Hash[
        r: (65535 - (c * (255 - k) + (k << 8))) >> 8,
        g: (65535 - (m * (255 - k) + (k << 8))) >> 8,
        b: (65535 - (y * (255 - k) + (k << 8))) >> 8
    ].map { |k, v| [k, clamp(v, 0, 255)] }
end

benchmark do
    cmyk_to_rgb(rand(255), rand(255), rand(255), rand(255))
end
def clamp(num, min, max)
    [min, num, max].sort[1]
end

def cmyk_to_rgb(c, m, y, k)
    Hash[
        r: (65535 - (c * (255 - k) + (k << 8))) >> 8,
        g: (65535 - (m * (255 - k) + (k << 8))) >> 8,
        b: (65535 - (y * (255 - k) + (k << 8))) >> 8
    ].map { |k, v| [k, clamp(v, 0, 255)] }
end

benchmark do
    cmyk_to_rgb(rand(255), rand(255), rand(255), rand(255))
end

Random inputs stop the whole thing being totally optimised away

Warms up and then reports iterations per second
clamp in Pure Ruby

- GraalVM: 10,000,000 Operations Per Second
- JRuby+invokedynamic: 0 Operations Per Second
- Ruby: 0 Operations Per Second
clamp in Pure Ruby

This is what GraalVM is giving you for Ruby before we even start talking about JavaScript
```ruby
require 'v8'

context = V8::Context.new

$clamp = context.eval("'function clamp(num, min, max) {
  if (num < min) {
    return min;
  } else if (num > max) {
    return max;
  } else {
    return num;
  }
  
  clamp;
}'")

def cmyk_to_rgb(c, m, y, k)
  Hash[
    r: (65535 - (c * (255 - k) + (k << 8))) >> 8,
    g: (65535 - (m * (255 - k) + (k << 8))) >> 8,
    b: (65535 - (y * (255 - k) + (k << 8))) >> 8
  ].map { |k, v| [k, $clamp.call(v, 0, 255)] }
end
```
require 'v8'

context = V8::Context.new

$clamp = context.eval("function clamp(num, min, max) {
    if (num < min) {
        return min;
    } else if (num > max) {
        return max;
    } else {
        return num;
    }
}
clamp;")

def cmyk_to_rgb(c, m, y, k)
  Hash[
    r: (65535 - (c * (255 - k) + (k << 8))) >> 8,
    g: (65535 - (m * (255 - k) + (k << 8))) >> 8,
    b: (65535 - (y * (255 - k) + (k << 8))) >> 8
  ].map { |k, v| [k, $clamp.call(v, 0, 255)] }
end

Not only have we rewritten in JavaScript, but the JavaScript code is simpler than the Ruby
clamp in Ruby and JavaScript with V8

- **Ruby (just Ruby)**
- **Ruby (Ruby + JS with V8)**

- Operations Per Second
  - 0
  - 50000
  - 100000
  - 150000
  - 200000
  - 250000
  - 300000
  - 350000
require 'rhino'

context = Rhino::Context.new
clamp in Ruby and JavaScript with JRuby and Rhino

Operations Per Second

<table>
<thead>
<tr>
<th>JRuby+indy (just Ruby)</th>
<th>JRuby+indy (Ruby + JS with Rhino)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
factory = javax.script.ScriptEngineManager.new
engine = factory.getEngineByName 'nashorn'
bindings = engine.createBindings

$clamp = engine.eval("function clamp(num, min, max) {
    if (num < min) {
        return min;
    } else if (num > max) {
        return max;
    } else {
        return num;
    }
}", bindings)

def cmyk_to_rgb(c, m, y, k)
    Hash[
        r: (65535 - (c * (255 - k) + (k << 8))) >> 8,
        g: (65535 - (m * (255 - k) + (k << 8))) >> 8,
        b: (65535 - (y * (255 - k) + (k << 8))) >> 8
    ].map { |k, v| [k, $clamp.call(v, 0, 255)] }
end
clamp in Ruby and JavaScript with JRuby and Nashorn

Operations Per Second

JRuby+indy (just Ruby)  JRuby+indy (Ruby + JS with Rhino)  JRuby+indy (Ruby + JS with Nashorn)
function clamp(num, min, max) {
    if (num < min) {
        return min;
    } else if (num > max) {
        return max;
    } else {
        return num;
    }
}

def cmyk_to_rgb(c, m, y, k)
    Hash[{ (r: (65535 - (c * (255 - k) + (k << 8))) >> 8,
    g: (65535 - (m * (255 - k) + (k << 8))) >> 8,
    b: (65535 - (y * (255 - k) + (k << 8))) >> 8
}).map { |k, v| [k, clamp(v, 0, 255)] }]
end
clamp in Ruby and JavaScript with GraalVM

- **GraalVM (just Ruby)**
- **GraalVM (Ruby + JS)**
- **JRuby+invokedynamic**
- **Ruby**
clamp in all configurations
clamp in all configurations

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Operations Per Second</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraalVM (just Ruby)</td>
<td>100000000</td>
</tr>
<tr>
<td>GraalVM (Ruby + JS)</td>
<td>100000000</td>
</tr>
<tr>
<td>JRuby+indy (just Ruby)</td>
<td>100000000</td>
</tr>
<tr>
<td>JRuby+indy (Ruby + JS with Rhino)</td>
<td>100000000</td>
</tr>
<tr>
<td>JRuby+indy (Ruby + JS with Nashorn)</td>
<td>100000000</td>
</tr>
<tr>
<td>Ruby (just Ruby)</td>
<td>100000000</td>
</tr>
<tr>
<td>Ruby (Ruby + JS with V8)</td>
<td>100000000</td>
</tr>
</tbody>
</table>
How Graal achieves this
Conventional JVM implementations of languages work by emitting JVM bytecode – the same thing that the Java compiler does.
Hotspot
Hotspot

JIT
Graal

Hotspot
Slightly confusing terminology...

• Graal is a new JIT compiler for the JVM

• Graal VM is the JVM, with Graal, Truffle, and our languages bundled in it

• Truffle uses Graal on your behalf
Guest Language

Bytecode

JVM
Guest Language

Compiler internal data structures, optimisation passes, machine code, ...

Graal
Guest Language

language interpreter

Truffle

Graal
The very basics of Truffle and Graal
\[ x + y \times z \]

load_local x
load_local y
load_local z
call *
call +

pushq %rbp
movq %rsp, %rbp
movq %rdi, -8(%rbp)
movq %rsi, -16(%rbp)
movq %rdx, -24(%rbp)
movq -16(%rbp), %rax
movl %eax, %edx
movq -24(%rbp), %rax
imull %edx, %eax
movq -8(%rbp), %rdx
addl %edx, %eax
popq %rbp
ret
\[ x + y \times z \]

```
load_local x
load_local y
load_local z
call *
call +
```

```
pushq %rbp
movq %rsp, %rbp
movq %rdi, -8(%rbp)
movq %rsi, -16(%rbp)
movq %rdx, -24(%rbp)
movq -16(%rbp), %rax
movl %eax, %edx
movq -24(%rbp), %rax
imull %edx, %eax
movq -8(%rbp), %rdx
addl %edx, %eax
popq %rbp
ret
```
\[ x + y \times z \]

```
load_local x
load_local y
load_local z
call *
call +
```

```
pushq %rbp
movq %rsp, %rbp
movq %rdi, -8(%rbp)
movq %rsi, -16(%rbp)
movq %rdx, -24(%rbp)
movq -16(%rbp), %rax
movl %eax, %edx
movq -24(%rbp), %rax
imull %edx, %eax
movq -8(%rbp), %rdx
addl %edx, %eax
popq %rbp
ret
```
Node Rewriting for Profiling Feedback

AST Interpreter
Uninitialized Nodes

Node Rewriting for Profiling Feedback

Node Transitions

Uninitialized Integer

String

Double

Generic

AST Interpreter Uninitialized Nodes

Node Rewriting for Profiling Feedback

AST Interpreter
Rewritten Nodes

Compilation using Partial Evaluation

Compiled Code

Uninitialized Integer
Generic Double
String

codon.com/compilers-for-free
Deoptimization to AST Interpreter

Node Rewriting to Update Profiling Feedback

Recompilation using Partial Evaluation

Frequently executed call
How effective is this in the extreme?
def sum(n):
    i = 0
    a = 0
    while i < n
        i += 1
        a += n
    end
    a
end

values = (1..100).to_a

loop do
    values.each do |v|
        sum(v)
    end
end

function sum(n) {
    var i = 0;
    var a = 0;
    while (i < n) {
        i += 1;
        a += n;
    }
    return a;
}

values = (1..100).to_a

loop do
    values.each do |v|
        sum(v)
    end
end
Looking at these loops here

def sum(n):
    i = 0
    a = 0
    while i < n:
        i += 1
        a += n
    return a

values = (1..100).to_a

loop do
    values.each do |v|
        sum(v)
    end
end

function sum(n) {
    var i = 0;
    var a = 0;
    while (i < n) {
        i += 1;
        a += n;
    }
    return a;
}

values = (1..100).to_a

loop do
    values.each do |v|
        sum(v)
    end
end
def sum(n):
    i = 0
    a = 0
    while i < n:
        i += 1

loop do
    values.each do |v|
        sum(v)
    end
end

function sum(n) {
    var i = 0;
    var a = 0;
    while (i < n) {
        i += 1;
        a += n
    }

loop do
    values.each do |v|
        sum(v)
    end
end
def add(a, b)
    a + b
end

def sum(n)
    i = 0
    a = 0
    while i < n
        i += 1
        a = add(a, n)
    end
    a
end

function add(a, b) {
    return a + b;
}

def sum(n)
    i = 0
    a = 0
    while i < n
        i += 1
        a = add(a, n)
    end
    a
end
def add(a, b)
    a + b
end

function add(a, b) {
    return a + b;
}

a = add(a, n)
end

a = add(a, n)
end
def add(a, b)
    a + b
end

function add(a, b) {
    return a + b;
}

0x0000000103a7dc70: mov  esi,edi
0x0000000103a7dc72: add  esi,r9d
0x0000000103a7dc75: jo  0x0000000103a7dda2
0x0000000103a7dc7b: inc  ecx
0x0000000103a7dc7d: mov  edi,esi
0x0000000103a7dc7f: cmp  r9d,ecx
0x0000000103a7dc82: jg  0x0000000103a7dc70

end
end
end
end
def add(a, b)
    a + b
end

function add(a, b) {
    return a + b;
}

0x0000000103a7dc70: mov    esi,edi
0x0000000103a7dc72: add    esi,r9d
0x0000000103a7dc75: jo     0x0000000103a7dda2  
0x0000000103a7dc7b: inc    ecx
0x0000000103a7dc7d: mov    edi,esi
0x0000000103a7dc7f: cmp    r9d,ecx
0x0000000103a7dc82: jg     0x0000000103a7dc70 

(a, n)

end

end
What is this for?
• We’re not really suggesting that people routinely write alternate methods in different languages
• We’re not really suggesting that people routinely write alternate methods in different languages

• More about removing the consideration of performance from the decision if you do want to combine languages
• Could make all library ecosystems available to all applications
• May be useful for unifying a front-end and back-end
• May be useful in handling legacy applications and incremental changes in implementation language
How to use GraalVM
GraalVM – everything in one package today

• Includes:
  – JVM (RE or DK)
  – Java
  – JavaScript
  – Ruby
  – R
  – More in the future

• Binary tarball release

• Mac or Linux
Java 9 – runs on an unmodified JVM

- JS
- R
- others...
- Truffle
- Graal
- Hotspot

Java

C++
Java 9 – runs on an unmodified JVM

JVMCI (JVM Compiler Interface)
Java 9 – runs on an unmodified JVM

- JS
- Ruby
- others...
- Truffle
- Graal
- Hotspot

via OTN, Maven etc

Java 9, JEP 243
Takeaways
• Oracle Labs is building Graal VM to support polyglot programs and programmers
• Extremely high performance for the languages on their own
• Completely unprecedented high performance for language interoperability
• Will work on an unmodified Java 9 JVM, or available as a bundle today
• Still at the research stage, but moving towards being something more than that
Where to find more information
Oracle Labs GraalVM and JVMCI JDK Downloads

Thank you for downloading this release of the Oracle Labs GraalVM. With this release, one can execute Java applications with Graal, as well as applications written in JavaScript, Ruby, and R, with our Polyglot language engines.

You must accept the OTN License Agreement to download this software.
- Accept License Agreement
- Decline License Agreement

- GraalVM preview for Linux (v0.16), Development Kit
- GraalVM preview for Linux (v0.16), Runtime Environment
- GraalVM preview for Mac OS X (v0.16), Development Kit
- GraalVM preview for Mac OS X (v0.16), Runtime Environment
- labjdk8u52-jvmci-0.20-darwin-amd64.tar.gz
- labjdk8u52-jvmci-0.20-linux-amd64.tar.gz
- labjdk8u52-jvmci-0.20-solaris-sparcv9.tar.gz

How to install GraalVM

Unpack the downloaded *.tar.gz file on your machine. You can then use the java executable to execute Java programs. All those executables are in the bin directory of GraalVM. You might want to add that directory to your operating system’s PATH.

More detailed getting started instructions are available in the README files in the download. The README files for the language engines can be found in /languages/.
@chrisgseaton
github.com/graalvm
gitter.im/graalvm/graal-core
Search ‘otn graalvm’
Acknowledgements

Oracle
Danilo Ansaloni
Stefan Anzinger
Cosmin Basca
Daniele Bonetta
Matthias Brantner
Petr Chalupa
Jürgen Christ
Laurent Daynès
Gilles Duboscq
Martin Entlicher
Brandon Fish
Bastian Hossbach
Christian Humer
Mick Jordan
Vojin Jovanovic
Peter Kessler
David Leopoldseder
Kevin Menard
Jakub Podlešák
Aleksandar Prokopec
Tom Rodriguez

Oracle (continued)
Roland Schatz
Chris Seaton
Doug Simon
Štěpán Šindelář
Zbyněk Šlajchrt
Lukas Stadler
Codrut Stancu
Jan Štola
Jaroslav Tulach
Michael Van De Vanter
Adam Welc
Christian Wimmer
Christian Wirth
Paul Wögerer
Mario Wölczko
Andreas Wöß
Thomas Würthinger

Oracle Interns
Brian Belleville
Miguel Garcia
Shams Imam
Alexey Karyakin
Stephen Kell
Andreas Kunft
Volker Lanting
Gero Leinemann
Julian Lettner
Joe Nash
David Piorkowski
Gregor Richards
Robert Seilbeck
Rifat Shariyar

Alumni
Erik Eckstein
Michael Haupt
Christos Kotselidis
Hyunjin Lee
David Leibs
Chris Thalinger
Till Westmann

 JKU Linz
Prof. Hanspeter Mössenböck
Benoit Daloze
Josef Eisl
Thomas Feichtinger
Matthias Grimmer
Christian Häubl
Josef Haider
Christian Huber
Stefan Marr
Manuel Rigger
Stefan Rumzucker
Bernhard Urban

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

LaBRI
Floréal Morandat

University of California, Irvine
Prof. Michael Franz
Gulfem Savrun Yeniceri
Wei Zhang

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

T. U. Dortmund
Prof. Peter Marwedel
Helena Kotthaus
Ingo Korb

University of California, Davis
Prof. Duncan Temple Lang
Nicholas Ulle

University of Lugano, Switzerland
Prof. Walter Binder
Sun Haiyang
Yudi Zheng
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.
Integrated Cloud
Applications & Platform Services