One VM to Rule Them All

Christian Wimmer, Chris Seaton

VM Research Group, Oracle Labs
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. 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.
One Language to Rule Them All?

Let's ask Google…

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 ...
One Language to Rule Them All?
Let’s ask Stack Overflow…

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
“Write Your Own Language”

**Current situation**

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

**How it should be**

1. Prototype a new language in Java
2. Parser and language work to build syntax tree (AST)
3. Execute using AST interpreter
4. People start using it
   - And it is already fast
Truffle Requirements

Simplicity + Generality + Performance

Ruby, JavaScript, Python, R, J, Java, Groovy, Clojure, Scala ...

@Specialization(rewriteOn=ArithmeticException.class)
int add(int l, int r) {
  return Math.addExact(l, r);
}

@Specialization
double add(double l, double r) {
  return l + r;
}

@Specialization(guards = "isString")
String doString(Object l, Object r) {
  return l.toString() + r.toString();
}

function f(a, n) {
  var x = 0;
  while (n-- > 0) {
    x = x + a[n];
  }
  return x;
}

L1: decl rax
jz L2
movl rcx, rdx[16+4*rax]
cvtisi2sd xmm1, rcx
addsd xmm0, xmm1
jmp L1
L2:
System Structure

Language agnostic dynamic compiler

Common API between language implementation and optimization system

Low-footprint VM, also suitable for embedding

Integrate with Java applications

Your language here!

Graal VM

Substrate VM

Truffle

Graal

Ruby

JavaScript

Python

R

...
Truffle Approach

Node Rewriting for Type Feedback

AST Interpreter
Uninitialized Nodes

AST Interpreter
Rewritten Nodes

Compiled Code

Automatic Partial Evaluation

Eliminate boxing of primitive values
Eliminate dynamic type checks
AST Inlining

Syntax tree nodes are “stable”
Aggressive constant folding, method inlining, escape analysis
Deoptimize compiled code on tree rewrite
More Details on Truffle
Accepted for Onward! 2013, October 26-31 2013, Indianapolis, IN

One VM to Rule Them All

Thomas Würthinger*  Christian Wimmer*  Andreas Woß†  Lukas Stadler†
Gilles Duboscq†  Christian Humer†  Gregor Richards§  Doug Simon*  Mario Wolczko*

*Oracle Labs  †Institute for System Software, Johannes Kepler University Linz, Austria  §S³ Lab, Purdue University
{thomas.wuerthinger, christian.wimmer, doug.simon, mario.wolczko}@oracle.com
{woess, stadler, duboscq, christian.humer}@ssw.jku.at  gr@purdue.edu

Abstract

Building high-performance virtual machines is a complex and expensive undertaking; many popular languages still have low-performance implementations. We describe a new approach to virtual machine (VM) construction that amortizes much of the effort in initial construction by allowing new languages to be implemented with modest additional effort. The approach relies on abstract syntax tree (AST) interpretation where a node can rewrite itself to a more specialized or more general node, together with an optimizing compiler that exploits the structure of the interpreter. The compiler uses speculative assumptions and deoptimization in order to produce efficient machine code. Our initial experience suggests that high performance is attainable while preserving a modular and layered architecture, and that new high-performance language implementations can be obtained by writing little more than a stylized interpreter.

as Microsoft’s Common Language Runtime, the VM of the .NET framework [43]. These implementations can be characterized in the following way:

- Their performance on typical applications is within a small integer multiple (1-3x) of the best statically compiled code for most equivalent programs written in an unsafe language such as C.
- They are usually written in an unsafe, systems programming language (C or C++).
- Their implementation is highly complex.
- They implement a single language, or provide a bytecode interface that preferentially advantages a narrow set of languages to the detriment of other languages.

In contrast, there are numerous languages that are popular, have been around for about 20 years, and yet still have
Ruby Prototype: High Performance

Fastest Ruby implementation ...

... for the few benchmarks that we looked at
Ruby Prototype: Low Footprint

Startup time ("Hello World") comparable to MRI
Ruby Prototype: Completeness

- RubySpec
  - A library of executable assertions that covers the language, core library and standard library
  - This is the defacto Ruby spec
  - Gives us a quantifiable result for how much of Ruby we implement correctly

Over 45% of RubySpec
Completeness
## Completeness: Informally

<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></td>
</tr>
<tr>
<td>callcc and Continuation</td>
<td>✓</td>
<td>Very limited support, the same as JRuby</td>
</tr>
<tr>
<td>Fibers</td>
<td>✓</td>
<td>Slightly 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></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></td>
</tr>
<tr>
<td>ObjectSpace</td>
<td>✓</td>
<td></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>

Completeness: More formally via RubySpec

Running language tests

- Topaz: 0.00%
- RubyTruffle: 10.00%
- JRuby: 100.00%
Low Footprint
Substrate VM Execution Model

Static Analysis

Ahead-of-Time Compilation

Java Application

JDK

Substrate VM

All Java classes from application, JDK, and Substrate VM

Reachable methods, fields, and classes

Application running without compilation or class loading

Initial Heap

Machine Code

OS
Startup Performance
Running "Hello World"

Execution time:
```
time -f "%e"
```

Memory footprint:
```
time -f "%M"
```

<table>
<thead>
<tr>
<th></th>
<th>MRI</th>
<th>JRuby</th>
<th>Truffle on JVM</th>
<th>Truffle on SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution</td>
<td>13</td>
<td>353</td>
<td>688</td>
<td>14</td>
</tr>
<tr>
<td>Memory</td>
<td>5</td>
<td>35</td>
<td>53</td>
<td>9</td>
</tr>
</tbody>
</table>

ORACLE
High Performance
Why is Ruby Slow?

\[-b + \frac{(\text{Math.sqrt}(b^{**2} - 4*a*c))}{2*a}\]
Why is Ruby Slow?

\[-b + (\text{Math.sqrt}(b^{2} - 4*a*c)) / 2*a\]

execute b
check that b is a Float
check that the negate method in Float has not changed
calculate negation
check the result of that is a Float
execute b
check that b is a Float
check that the power method in Float has not changed
calculate power
check the result of that is a Float
execute a
check that a is a Float
check that the multiply method in Float has not changed
calculate multiplication
check the result of that is a Float
execute c
check that c is a Float
check that the multiply method in Float has not changed
calculate multiplication
check the result of that is a Float
check that Math has not changed
check that the sqrt method in Math has not changed
calculate sqrt
check the result of that is a Float
execute a
check that a is a Float
check that the multiply method in Float has not changed
calculate multiplication
check the result of that is a Float
check that the division method in Float has not changed
calculate division
Improving Performance Using Truffle

-b + (Math.sqrt(b**2 - 4*a*c)) / 2*a

execute b
check that b is a Float
check that the negate method in Float has not changed
calculate negation
check the result of that is a Float
execute b
check that b is a Float
check that the power method in Float has not changed
calculate power
check the result of that is a Float
execute a
check that a is a Float
check that the multiply method in Float has not changed
calculate multiplication
check the result of that is a Float
execute c
check that c is a Float
check that the multiply method in Float has not changed
calculate multiplication
check the result of that is a Float
check that Math has not changed
check that the sqrt method in Math has not changed
calculate sqrt
check the result of that is a Float
execute a
check that a is a Float
check that the multiply method in Float has not changed
calculate multiplication
check the result of that is a Float
check that the division method in Float has not changed
calculate division
-b + (Math.sqrt(b**2 - 4*a*c)) / 2*a

execute b
check that b is a Float
check that the negate method in Float has not changed
calculate negation
check the result of that is a Float
execute b
check that b is a Float
check that the power method in Float has not changed
calculate power
check the result of that is a Float
execute a
check that a is a Float
check that the multiply method in Float has not changed
calculate multiplication
check the result of that is a Float
execute c
check that c is a Float
check that the multiply method in Float has not changed
calculate multiplication
check the result of that is a Float
check that Math has not changed
check that the sqrt method in Math has not changed
calculate sqrt
check the result of that is a Float
execute a
check that a is a Float
check that the multiply method in Float has not changed
calculate multiplication
check the result of that is a Float
check that the division method in Float has not changed
calculate division
-b + (Math.sqrt(b**2 - 4*a*c)) / 2*a

execute b
check that b is a Float
check that the negate method in Float has not changed
calculate negation
check the result of that is a Float
execute b
check that b is a Float
check that the power method in Float has not changed
calculate power
check the result of that is a Float
execute a
check that a is a Float
check that the multiply method in Float has not changed
calculate multiplication
check the result of that is a Float
execute c
check that c is a Float
check that the multiply method in Float has not changed
calculate multiplication
check the result of that is a Float
check that Math has not changed
check that the sqrt method in Math has not changed
calculate sqrt
check the result of that is a Float
execute a
check that a is a Float
check that the multiply method in Float has not changed
calculate multiplication
check the result of that is a Float
check that the division method in Float has not changed
calculate division
Improving Performance Using Truffle

\[-b + (\text{Math.sqrt}(b^{**2} - 4*a*c)) / 2*a\]

execute b
check that b is a Float
check that the negate method in Float has not changed

calculate negation
check the result of that is a Float
execute b
check that b is a Float
check that the power method in Float has not changed

calculate power
check the result of that is a Float
execute a
check that a is a Float
check that the multiply method in Float has not changed

calculate multiplication
check the result of that is a Float
execute c
check that c is a Float
check that the multiply method in Float has not changed

calculate multiplication
check the result of that is a Float
check that Math has not changed
check that the sqrt method in Math has not changed

calculate sqrt
check the result of that is a Float
execute a
check that a is a Float
check that the multiply method in Float has not changed

calculate multiplication
check the result of that is a Float
check that the division method in Float has not changed

calculate division
-b + (Math.sqrt(b**2 - 4*a*c)) / 2*a

execute b
check that b is a Float
check that the negate method in Float has not changed
calculate negation
check the result of that is a Float
execute b
check that b is a Float
check that the power method in Float has not changed
calculate power
check the result of that is a Float
execute a
check that a is a Float
check that the multiply method in Float has not changed
calculate multiplication
check the result of that is a Float
execute c
check that c is a Float
check that the multiply method in Float has not changed
calculate multiplication
check the result of that is a Float
check that Math has not changed
check that the sqrt method in Math has not changed
calculate sqrt
check the result of that is a Float
execute a
check that a is a Float
check that the multiply method in Float has not changed
calculate multiplication
check the result of that is a Float
check that the division method in Float has not changed
calculate division
Improving Performance Using Truffle

\[-b + (\text{Math.sqrt}(b^{**}2 - 4*a*c)) / 2*a\]

execute b
check that b is a Float
check that the negate method in Float has not changed
calculate negation
check the result of that is a Float
execute b
check that b is a Float
check that the power method in Float has not changed
calculate power
check the result of that is a Float
execute a
check that a is a Float
check that the multiply method in Float has not changed
calculate multiplication
check the result of that is a Float
execute c
check that c is a Float
check that the multiply method in Float has not changed
calculate multiplication
check the result of that is a Float
check that Math has not changed
check that the sqrt method in Math has not changed
calculate sqrt
check the result of that is a Float
execute a
check that a is a Float
check that the multiply method in Float has not changed
calculate multiplication
check the result of that is a Float
check that the division method in Float has not changed
calculate division
Improving Performance Using Truffle

-b + (Math.sqrt(b**2 - 4*a*c)) / 2*a

execute b
check that b is a Float
check that the negate method in Float has not changed
calculate negation
check the result of that is a Float
execute b
check that b is a Float
check that the power method in Float has not changed
calculate power
check the result of that is a Float
execute a
check that a is a Float
check that the multiply method in Float has not changed
calculate multiplication
check the result of that is a Float
execute c
check that c is a Float
check that the multiply method in Float has not changed
calculate multiplication
check the result of that is a Float
check that Math has not changed
calculate sqrt
check the result of that is a Float
execute a
check that a is a Float
check that the multiply method in Float has not changed
calculate multiplication
check the result of that is a Float
check that the division method in Float has not changed
calculate division
Improving Performance Using Graal

\[-b + (\text{Math.sqrt}(b^{**2} - 4*a*c)) / 2*a\]

execute b
check that the negate method in Float has not changed
calculate negation
execute b
check that the power method in Float has not changed
calculate power
execute a
check that the multiply method in Float has not changed
calculate multiplication
execute c
check that the multiply method in Float has not changed
calculate multiplication
check that Math has not changed
calculate \text{sqrt}
execute a
check that the multiply method in Float has not changed
calculate multiplication
check that the division method in Float has not changed
calculate division
Improving Performance Using Graal

\[-b + (Math.sqrt(b^{2} - 4*a*c)) / 2*a\]

execute b
check that the negate method in Float has not changed
calculate negation
execute b
check that the power method in Float has not changed
calculate power
execute a
check that the multiply method in Float has not changed
calculate multiplication
execute c
check that the multiply method in Float has not changed
calculate multiplication
check that Math has not changed
calculate sqrt
execute a
check that the multiply method in Float has not changed
calculate multiplication
check that the division method in Float has not changed
calculate division
-b + (Math.sqrt(b**2 - 4*a*c)) / 2*a

evaluate b
check that the negate method in Float has not changed
calculate negation
execute b
check that the power method in Float has not changed
calculate power
execute a
check that the multiply method in Float has not changed
calculate multiplication
execute c
check that the multiply method in Float has not changed
calculate multiplication
check that Math has not changed
calculate sqrt
execute a
check that the multiply method in Float has not changed
calculate multiplication
check that the division method in Float has not changed
calculate division

class Float
modified?

module Math
modified?
-b + (Math.sqrt(b**2 - 4*a*c)) / 2*a
Improving Performance Using Graal

\[-b + (\text{Math.sqrt}(b^2 - 4*a*c)) / 2*a\]

Java object `InstalledCode`

- execute `b`
- check that the negate method in Float has not changed
- calculate negation
- execute `b`
- check that the power method in Float has not changed
- calculate power
- execute `a`
- check that the multiply method in Float has not changed
- calculate multiplication
- execute `c`
- check that the multiply method in Float has not changed
- calculate multiplication
- check that Math has not changed
- check that the `sqrt` method in Math has not changed
- calculate `sqrt`
- execute `a`
- check that the multiply method in Float has not changed
- calculate multiplication
- check that the division method in Float has not changed
- calculate division

Class `Float`

- `modified?`

Module `Math`

- `modified?`

`.invalidate()`
Improving Performance Using Graal

unmodified = new Assumption();
unmodified.check();
unmodified.invalidate();
Improving Performance Using Graal

\[-b + (\text{Math.sqrt}(b^{**2} - 4*a*c)) / 2*a\]
Improving Performance

\[-b + \left(\text{Math.sqrt}(b^{**2} - 4*a*c)\right) / 2*a\]

execute b
calculate negation
execute b
calculate power
execute a
calculate multiplication
execute c
calculate multiplication
calculate sqrt
calculate multiplication
calculate division
Peak Performance

Speedup Relative to 1.8.7-p374

- Fannkuch
- N-Body
Peak Performance

Speedup Relative to jruby-1.7.4-server-invokedynamic

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Speedup Relative to jruby-1.7.4-server-invokedynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>jruby-1.7.4-server</td>
<td>Fannkuch: 0.0, N-Body: 0.0</td>
</tr>
<tr>
<td>jruby-1.7.4-server-invokedynamic</td>
<td>Fannkuch: 1.2, N-Body: 1.2</td>
</tr>
<tr>
<td>topaz-dev</td>
<td>Fannkuch: 4.0, N-Body: 4.0</td>
</tr>
<tr>
<td>truffle-server-nograal</td>
<td>Fannkuch: 0.0, N-Body: 0.0</td>
</tr>
<tr>
<td>truffle-server</td>
<td>Fannkuch: 5.0, N-Body: 5.0</td>
</tr>
</tbody>
</table>
Simplicity

- One intern working for five months on the Ruby implementation
- New to Truffle, Graal and Ruby

- Written using Eclipse
- Debugged as a normal Java program using the server compiler
- Run using Graal for testing and performance numbers

- No mention in the implementation of bytecode, classloaders, assembly, system calls, OSR
- One very minor use of Unsafe, one very minor use of reflection
Acknowledgments

Oracle Labs
Laurent Daynès
Michael Haupt
Peter Kessler
Christos Kotselidis
David Leibs
Roland Schatz
Chris Seaton
Doug Simon
Michael Van De Vanter
Christian Wimmer
Christian Wirth
Mario Wolczko
Thomas Würthinger
Laura Hill (Manager)

Oracle Labs Interns
Danilo Ansaloni
Daniele Bonetta
Shams Imam
Stephen Kell
Helena Kotthaus
Gregor Richards
Rifat Shariyar
Codrut Stancu
Wei Zhang

JKU Linz
Gilles Duboscq
Matthias Grimmer
Christian Häubl
Josef Haider
Christian Humer
Christian Huber
Manuel Rigger
Lukas Stadler
Bernhard Urban
Andreas Wöß

Purdue University
Tomas Kalibera
Floreal Morandat
Petr Maj
Prof. Jan Vitek

University of California, Irvine

University of Dortmund

Prof. Hanspeter Mössenböck
Hardware and Software

Engineered to Work Together
System Structure (Details)

- **Guest Language Application**
- **Guest Language Implementation**
  - Language Parser
  - AST Interpreter
- **Truffle API**
  - Framework for Node Rewriting
- **Truffle Optimizer**
  - Partial Evaluation using Graal
- **VM Runtime Services**
  - Garbage Collector
  - Stack Walking
  - Graal Compiler
  - Deoptimization

**AOT Optimization:** using Graal for static analysis and AOT compilation

- **Hosted on any Java VM**
  - (slow, for guest language development and debugging only)
- **Hosted on Graal VM**
  - (fast, for integration of guest language code with existing Java applications)