The TruffleRuby Compilation Pipeline

Just-in-time compiling Ruby with self-specializing ASTs and partial evaluation

Chris Seaton
Research Manager
Oracle Labs
March 2019
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.
TruffleRuby basics
We wanted to build a Ruby that

• Runs idiomatic Ruby code faster
• Runs Ruby code in parallel
• Executes C extensions in a managed environment
• Adds fast and low-overhead interoperability with other languages
• Provides new tooling such as debuggers and monitoring
• Has very high compatibility with the standard implementation of Ruby
$ rbenv install truffleruby-1.0.0-rc14
$ rbenv shell truffleruby-1.0.0-rc14
$ ruby -v
truffleruby 1.0.0-rc14, like ruby 2.6.2, GraalVM CE Native [x86_64-darwin]
$
```ruby
require 'erb'

template = ERB.new(%{
  <h1>Hello world!</h1>
  <p>The time is <%= now %></p>
})

loop do
  start = Time.now

  100_000.times do
    now = Time.now
    puts template.result(binding)
  end

  $stderr.puts Time.now - start
end
```
$ rbenv shell 2.6.2
$ ruby perf.rb > /dev/null
2.165717
2.134813
2.150839
2.139991
2.145957
2.17635
2.209064
2.196461
2.183587
$ rbenv shell 2.6.2
$ ruby --jit perf.rb > /dev/null
2.404086
2.436718
2.618684
2.391098
2.47526
2.488115
2.427324
2.407296
2.503696
$ rbenv shell jruby-9.2.6.0
$ ruby perf.rb > /dev/null
4.098946
3.117474
3.164635
3.18634
3.37483
3.305855
3.40638
3.088724
3.0775989999999998

$ rbenv shell jruby-9.2.6.0
$ ruby -Xcompile.invokedynamic=true perf.rb > /dev/null
3.8451880000000003
2.939869
3.010014
3.177851
3.1914369999999996
3.366082
2.927472
2.95315
2.990667
$ rbenv shell truffle ruby-1.0.0-rc14
$ ruby perf.rb > /dev/null
1.577
1.912
0.781
0.464
0.465
0.6
0.465
0.47
0.461
Current situation

- 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

How it should be

- 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
Automatic transformation of interpreters to compiler

GraalVM™

Embeddable in native or managed applications

OpenJDK™, node, ORACLE® Database, MySQL™, standalone
Compilation basics
def fib(n)
    if n <= 2
        1
    else
        fib(n - 1) + fib(n - 2)
    end
end

0x0000000011a63b6e4:  cmp  r10d, 0x3
0x0000000011a63b6e8:  jl   0x0000000011a63b6e8
0x0000000011a63b703:  sub  r13d, 0x1
0x0000000011a63b707:  jo   0x0000000011a63be54
0x0000000011a63b897:  call  0x0000000011950bde0
0x0000000011a63b8a8:  sub  ebx, 0x2
0x0000000011a63b8ab:  jo   0x0000000011a63da2
0x0000000011a63b9af:  call  0x0000000011950bde0
0x0000000011a63b9c5:  add  r13d,eax
0x0000000011a63bc8:  jo   0x0000000011a63bd7b
0x0000000011a63ba35:  test  DWORD PTR [rip+0xffffffff0e5e5cb],eax
0x0000000011a63ba3e:  ret
0x0000000011a63ba3f:  movabs rax, 0x6c012dd20
0x0000000011a63ba58:  test  DWORD PTR [rip+0xffffffff0e5e5a8],eax
0x0000000011a63ba61:  ret
def compile(ruby_source : String) : Array(UInt8)

...
Compiling Ruby
So You Want To Optimize Ruby

I was recently asked for a list of "hard problems" a Ruby implementation really needs to solve before reporting benchmark numbers. You know...the sort of problems that might invalidate early perf numbers because they impact how you optimize Ruby. This post is a rework of my response...I hope you find it informative!

Fixnum to Bignum promotion
In Ruby, Fixnum math can promote to Bignum when the result is out of Fixnum's range. On implementations that use tagged pointers to represent Fixnum (MRI, Rubinius, MacRuby), the Fixnum range is somewhat less than the base CPU bits (32/64). On JRuby, Fixnum is always a straight 64-bit signed value.

This promotion is a performance concern for a couple reasons:

- Every math operation that returns a new Fixnum must be range-checked. This slows all Fixnum operations.
- It is difficult (if not impossible) to predict whether a Fixnum math operation will return a Fixnum or a Bignum. Since Bignum is always represented as a full object (not a primitive or a tagged pointer) this impacts optimizing Fixnum math call sites.

Floating-point performance
A similar concern is the performance of floating point values. Most of the native implementations have tagged values for Fixnum but only one I know of (Macruby) uses tagged values for Float. This can
So you want to optimize Ruby?

- Fixnum to Bignum promotion
- Floating-point performance
- Closures
- Bindings and eval
- `callcc` and Continuation
- Fiber implementation
- Thread/frame/etc local $globals
- C extension support

- Ruby 1.9 encoding support
- Garbage collection and object allocation
- Concurrency / Parallelism
- Tracing/debugging
- ObjectSpace
- Method invalidation
- Constant lookup and invalidation
- Rails
The TruffleRuby compilation pipeline
```ruby
def fib(n)
  if n <= 2
    1
  else
    fib(n - 1) + fib(n - 2)
  end
end

loop do
  puts fib(30)
end
```
```bash
$ rbenv shell 2.6.2
$ ruby --dump=parse fib.rb

# Do NOT use this node dump for any purpose other than debug and research. Compatibility is not guaranteed.

# @ NODE_SCOPE (line: 1, location: (1,0)-(11,3))
#    nd_tbl: (empty)
#    nd_args:
#      (null node)
#    nd_body:
#      @ NODE_BLOCK (line: 1, location: (1,0)-(11,3))
#        nd_head (1):
#          @ NODE_DEFN (line: 1, location: (1,0)-(7,3))

#   nd_args:
#     @ NODE_ARRAY (line: 10, location: (10,7)-(10,14))
#       nd_alen: 1
#       nd_head:
#         @ NODE_FCALL (line: 10, location: (10,7)-(10,14))
#           nd_mid: :fib
#             nd_args:
#               @ NODE_ARRAY (line: 10, location: (10,11)-(10,13))
#                 nd_alen: 1
#                 nd_head:
#                   @ NODE_LIT (line: 10, location: (10,11)-(10,13))
#                     nd_lit: 30
#                     nd_next: (null node)

$
def :foo [:n]

if

send :<=

1

:n 2

send :-

:n 1

send :-

:n 2

send :+

2

1
def :foo [:n]

If

send :<=

1

:n

2

send :

send :+

send :-

:n

1

:n

2
def :foo [[:n]]

if

send :<=

1

:n 2

send :

:n 1

send :

:n 2

send :+

send :-

:n 1

send :-

:n 2
def :foo [:n]

:n must be Fixnum

if

send :<=

Fixnum#<=

send :+

Fixnum#+

send :-

Fixnum#-

:n 2

1 2

:n 1

:n 2
`def :foo [:n]
:n must be Fixnum

if Fixnum
Fixnum#<=Fixnum#
Fixnum#-Fixnum#
Fixnum#+

send :<=
:n 2

send :-
:n 1

send :-
:n 2

send :+: Fixnum#+
def :foo [:n]

1

if

send :<=

:n 2

send :+

send :-

:n 1

send :-

:n 2
```java
public class Fib {
    private static int fib(int n) {
        if (n <= 2) {
            return 1;
        } else {
            return fib(n - 1) + fib(n - 3);
        }
    }

    public static void main(String[] args) {
        while (true) {
            System.out.println(fib(30));
        }
    }
}
```
Back to Ruby...
Become registers

Become machine instructions
B0 -> B1, B2 [-1, -1]

_f tid d _instruction__org--------------------------------------------- (HIR)
# v0 d StartNode stateAfter: - #next: v8
~ i1 d Parameter
~ i4 d Constant
~ v5 d <@ .graalvm.compiler.nodes.calc.IntegerLessThanNode@ x: i1 y: i4
* v8 d If condition: v5 #trueSuccessor: v6 #falseSuccessor: v7

B1 <- B0 [-1, -1]

_f tid d _instruction__org--------------------------------------------- (HIR)
# v6 d Begin #next: v10
~ i9 d Constant
* v10 d Return result: i9 memoryMap: -

B2 <- B0 [-1, -1]

_f tid d _instruction__org--------------------------------------------- (HIR)
# v7 d Begin #next: i14
~ i11 d Constant
~ i12 d + x: i1 y: i11
v23 d HotSpotDirectCallTarget arguments: i12
?25 d FrameState outerFrameState: - values: i1
# i14 d Invoke stateAfter: - classInit: - callTarget: v23 stateDuring: ?25 #next: i19
~ i16 d Constant
~ i17 d + x: i1 y: i16
v24 d HotSpotDirectCallTarget arguments: i17
?26 d FrameState outerFrameState: - values: - i14
# i19 d Invoke stateAfter: - classInit: - callTarget: v24 stateDuring: ?26 #next: v22
~ i21 d + x: i14 y: i19
* v22 d Return result: i21 memoryMap: -
@Override
public void generate(NodeLIRBuilderTool generator) {
    generator.setResult(this, generateArithmetic(generator));
    generator.emitOverflowCheckBranch(getOverflowSuccessor(), getNext(), stamp, probability(getOverflowSuccessor()));
}

public final void jcc(ConditionFlag cc, Label l) {
    assert (0 <= cc.getValue()) && (cc.getValue() < 16) : "illegal cc";
    if (!isBound()) {
        jcc(cc, l.position(), false);
    } else {
        // Note: could eliminate cond. jumps to this jump if condition
        // is the same however, seems to be rather unlikely case.
        // Note: use jccb() if label to be bound is very close to get
        // an 8-bit displacement
        l.addPatchAt(position());
        emitByte(0x0F);
        emitByte(0x80 | cc.getValue());
        emitInt(0);
    }
}
(1) \( v1 = 10 \)
(2) \( v2 = 20 \)
(3) \( v3 = v1 + v2 \)
(4) \( v4 = v2 + v3 \)
(5) \( v1 = v3 + v4 \)
(6) \( v5 = v4 + v1 \)
(7) return v1 + v5

Linear Scan Register Allocation for the Java HotSpot™ Client Compiler

Christian Wimmer
0x000000011a63b6e4: cmp r10d, 0x3
0x000000011a63b6e8: jl 0x000000011a63ba3f
0x000000011a63b703: sub r13d, 0x1
0x000000011a63b707: jo 0x000000011a63be54
0x000000011a63b897: call 0x000000011950bde0
0x000000011a63b8a8: sub ebx, 0x2
0x000000011a63b8ab: jo 0x000000011a63bda2
0x000000011a63b9af: call 0x000000011950bde0
0x000000011a63b9c5: add r13d, eax
0x000000011a63b9c8: jo 0x000000011a63bd7b
0x000000011a63ba35: test DWORD PTR [rip+0xffffffff0e5e5cb], eax
0x000000011a63ba3e: ret
0x000000011a63ba3f: movabs rax, 0x6c012dd20
0x000000011a63ba58: test DWORD PTR [rip+0xffffffff0e5e5a8], eax
0x000000011a63ba61: ret
There’s more to TruffleRuby
ruby.rb

extension.c

Ruby Interpreter

C Interpreter
Automatic transformation of interpreters to compiler

GraalVM™

Embeddable in native or managed applications

OpenJDK™ node ORACLE® Database MySQL™

standalone

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.
Summing up
$ rbenv install truffleruby-1.0.0-rc14
$ rbenv shell truffleruby-1.0.0-rc14
$ ruby -v
truffleruby 1.0.0-rc14, like ruby 2.6.2, GraalVM CE Native [x86_64-darwin]
$
GraalVM is a universal virtual machine for running applications written in JavaScript, Python, Ruby, R, JVM-based languages like Java, Scala, Kotlin, Clojure, and LLVM-based languages such as C and C++.

GraalVM removes the isolation between programming languages and enables interoperability in a shared
Team

Oracle
Florian Angerer
Danilo Ansaloni
Stefan Anzinger
Martin Balin
Cosmin Basca
Daniele Bonetta
Dušan Bálek
Matthias Brantner
Lucas Braun
Petr Chalupa
Jürgen Christ
Laurent Daynès
Gilles Duboscq
Svatopluk Dědic
Martin Entlicher
Pit Fender
François Farquet
Brandon Fish
Mathias Gmitter
Christian Häubl
Peter Hofer
Bastian Hossbach
Christian Humer
Tomáš Hůrka
Mick Jordan

Oracle (continued)
Vojin Jovanovic
Ananthan Kandukuri
Harshad Kasture
Cansu Kaynak
Peter Kessler
Duncan MacGregor
Jiří Maršík
Kevin Menard
Milošlav Metelka
Tomáš Myšlík
Petr Pišl
Oleg Pliss
Jakub Podlešák
Aleksandar Prokopec
Tom Rodrigues
Roland Schatz
Benjamin Schlegel
Chris Seaton
Jiří Sedláček
Doug Simon
Štěpán Sindelář
Zbyněk Slajcht
Boris Spasojevic
Lukás Stadler
Codrut Stancu

Oracle (continued)
Jan Štola
Tomáš Stupka
Farhan Tauka
Jaroslav Tulach
Alexander Ulrich
Michael Van De Vanter
Aleksandar Vitorovic
Christian Wimmer
Christian Wirth
Paul Wögerer
Mario Wolczko
Andreas Woß
Thomas Würthinger
Tomáš Zezula
Yudi Zheng

Oracle Interns
Brian Belleville
Ondřej Douda
Juan Fumero
Miguel Garcia
Hugo Gruirox
Shams Imam
Berkin Ilbeyi
Hugo Kapp
Alexey Karyakin
Stephen Kell
Andreas Kunft
Volker Lanting
Gero Leinemann
Julian Lettner
Joe Nash
Tristan Overney
Aleksandar Pejovic
David Piorkowski
Philipp Riedmann
Gregor Richards
Robert Seibbeck
Rifat Sharayev

Red Hat
Andrew Dinn
Andrew Haley

Intel
Michael Berg

Twitter
Chris Thalinger

Oracle Alumni
Erik Eckstein
Michael Haupt
Christos Kotselidis
David Leibs
Adam Welc
Till Westermann

JGU Linz
Hanspeter Mössenböck
Bennot Daloze
Josef Eisl
Thomas Feichtinger
Josef Haider
Christian Huber
David Leopoldseder
Stefan Marr
Manuel Rigger
Stefan Rumzucker
Bernhard Urban

TU Berlin:
Volker Markl
Andreas Kunft
Jens Meiners
Tilmann Rabl

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

LaBRI
Floréal Morandat

Purdue University
Jan Vitek
Tomáš Kalibera
Petr Maj
Lei Zhao

T. U. Dortmund
Peter Marwedel
Helena Kotthaus
Ingo Korb

University of California, Davis
Duncan Temple Lang
Nicholas Ullie

University of California, Irvine
Michael Franz
Yeoul Na
Mohaned Qunaibit
Gulfem Savrun Yeniceri
Wei Zhang

University of California, Davis
Duncan Temple Lang
Nicholas Ullie

University of Lugano, Switzerland
Walter Binder
Sun Haiyang
Questions
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