Using LLVM and Sulong for Language C Extensions

Chris Seaton
Research Manager
VM Research Group
Oracle Labs

Manuel Rigger, Matthias Grimmer
JKU Linz
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.
Who we are and what we’re doing
The Ruby Logo is Copyright (c) 2006, Yukihiro Matsumoto. It is licensed under the terms of the Creative Commons Attribution-ShareAlike 2.5 agreement.

JS Logo Copyright (c) 2011 Christopher Williams <chris@iterativedesigns.com>, MIT licence.

You can distribute the logo under the terms of the Creative Commons Attribution-ShareAlike 4.0 International license (CC-BY-SA 4.0) or (at your option) the GNU General Public License version 2 (GPL-2).
Language C extensions
```python
def clamp(num, min, max):
    [min, num, max].sort()[1]
end
```
VALUE psd_native_util_clamp(VALUE self, 
   VALUE r_num, VALUE r_min, VALUE r_max) {
  int num = FIX2INT(r_num);
  int min = FIX2INT(r_min);
  int max = FIX2INT(r_max);

  return num > max ? r_max : (num < min ? r_min : r_num);
}
Performance on Ruby C Extensions Oily PNG and PSD Native

The C extension problem


```c
struct RString {
    struct RBasic basic;
    union {
        struct {
            long len;
            char *ptr;
            union {
                long capa;
                VALUE shared;
            } aux;
        } heap;
        char ary[RSTRING_EMBED_LEN_MAX + 1];
    } as;
};
```
```c
static VALUE
ossl_rand_bytes(VALUE self, VALUE len)
{
    VALUE str;
    int n = NUM2INT(len);
    int ret;

    str = rb_str_new(0, n);
    ret = RAND_bytes((unsigned char *)RSTRING_PTR(str), n);
    if (ret == 0) {
        ossl_raise(eRandomError, "RAND_bytes");
    } else if (ret == -1) {
        ossl_raise(eRandomError, "RAND_bytes is not supported");
    }

    return str;
}
```
Values need to be converted as they go from Ruby to native.

Call from Ruby to native is extremely hot.

```
(0...@num_pixels).step(pixel_step) do |i|
    ...
    rgb = PSD::Color.cmyk_to_rgb(255 - c, 255 - m, 255 - y, 255 - k)
    ...

    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, Util.clamp(v, 0, 255)] }
    end
```

```
VALUE psd_native_util_clamp(VALUE self,
                               VALUE r_num, VALUE r_min, VALUE r_max) {
    int num = FIX2INT(r_num);
    int min = FIX2INT(r_min);
    int max = FIX2INT(r_max);

    return num > max ? r_max : (num < min ? r_min : r_num);
}
```
Array implementation
pointer taken and stored for later

```c
VALUE* bg_pixels = RARRAY_PTR(rb_funcall(self, rb_intern("pixels"), 0));
VALUE* fg_pixels = RARRAY_PTR(rb_funcall(other, rb_intern("pixels"), 0));

long x = 0;
long y = 0;
for( y = 0; y < other_height; y++ ){
    for( x = 0; x < other_width; x++ ){
        bg_index = ( x + offset_x ) + ( y + offset_y ) * self_width;
        bg_pixels[bg_index] = UINT2NUM(
            oily_png_compose_color(
                NUM2UINT( fg_pixels[x+y*other_width] ),
                NUM2UINT( bg_pixels[bg_index] )
            )
        );
    }
}
```

When they’re used there’s no indication someone else is managing them
Previous solutions
bool RubyString::jsync(JNIEnv* env)
{
    if (radata.readonly && radata.rstring != NULL) {
        // Don't sync anything, just clear the cached data
        radata.rstring = NULL;
        radata.readonly = false;
        return false;
    }

    if (radata.rstring != NULL && radata.rstring->ptr != NULL) {
        jobject byteList = env->GetObjectField(obj, RubyString_value_field);
        jobject bytes = env->GetObjectField(byteList, ByteList_bytes_field);
        jint begin = env->GetIntField(byteList, ByteList_begin_field);
        checkExceptions(env);
        env->DeleteLocalRef(byteList);

        RString* rstring = radata.rstring;
        env->GetByteArrayRegion(jbyteArray) bytes, begin, rstring->len,
        (jbyte *) rstring->ptr;
        checkExceptions(env);
        env->GetIntField(byteList, ByteList_length_field, rstring->len);
        env->DeleteLocalRef(bytes);
    }

    return true;
}

bool RubyString::nsync(JNIEnv* env)
{
    jobject byteList = env->GetObjectField(obj, RubyString_value_field);
    checkExceptions(env);
    jobject bytes = env->GetObjectField(byteList, ByteList_bytes_field);
    checkExceptions(env);
    jint begin = env->GetIntField(byteList, ByteList_begin_field);
    checkExceptions(env);
    jint length = env->GetIntField(byteList, ByteList_length_field);
    checkExceptions(env);
    jint capacity = env->GetArrayLength((jarray) bytes) - begin;
    checkExceptions(env);
    env->DeleteLocalRef(byteList);

    RString* rstring = radata.rstring;

    if ((capacity > rstring->capa) || (rstring->capa == 0)) {
        rstring->capa = capacity;
        rstring->ptr = (char *) realloc(rstring->ptr, rstring->capa + 1);
    }

    env->GetByteArrayRegion((jbyteArray) bytes, begin, length,
    (jbyte *) rstring->ptr);
    checkExceptions(env);
    env->DeleteLocalRef(bytes);

    rstring->ptr[rstring->len = length] = 0;
    return true;
}
Our new solution
• Interpret both the Ruby and the C
• Actually, interpret the LLVM IR of the C to simplify
• JIT compile the Ruby and the C
• Use a single high and low level IR for both
• Forget which language the IR came from and optimise them together
• Give virtualised pointers to the C program
How Sulong and JRuby+Truffle work
Hotspot
Hotspot

JIT
Truffle

JIT

Hotspot

JIT
Node Rewriting for Profiling Feedback

AST Interpreter

Uninitialized Nodes

Compilation using Partial Evaluation

Compiled Code

Node Transitions

Node Rewriting for Profiling Feedback

Node Transitions

AST Interpreter
Uninitialized Nodes

Node Rewriting for Profiling Feedback

AST Interpreter
Rewritten Nodes

Compilation using Partial Evaluation

Compiled Code

codon.com/compilers-for-free
Node Rewriting for Profiling Feedback

Compilation using Partial Evaluation

Node Rewriting to Update Profiling Feedback

Recompilation using Partial Evaluation

Deoptimization to AST Interpreter

Node Rewriting to Update Profiling Feedback

Recompilation using Partial Evaluation

BigInteger → double → int
BigInteger
JVMCI (JVM Compiler Interface)

- Hotspot
- Graal
- Truffle
- JS
- R
- Ruby

Languages:
- Java
- C++
Hotspot

Graal

Truffle

JS  R  Ruby

via Maven etc

Java 9
Completeness – language and core library

99%  Ruby language
    JRuby passes 94%

95%  Ruby core library
    JRuby passes 95%
Completeness – the basic Rails stack

- Active Support
- Active Model
  - Active Record
- Action View
- Action Pack
  - Action Mailer
  - Railties
  - Sprockets-Rails
- Active Job
- Spring

Basic functionality works

100% 100% 98% 37%
Classic research benchmarks – 10-20x faster

![Graph showing speedup compared to Ruby]

**Benchmarks bound by allocation or BigInteger performance**
VALUE psd_native_util Clamp(VALUE self,
    VALUE r_num, VALUE r_min, VALUE r_max) {
    int num = FIX2INT(r_num);
    int min = FIX2INT(r_min);
    int max = FIX2INT(r_max);

    return num > max ? r_max : (num < min ? r_min : r_num);
}
define i8* @psd_native_util_clamp(i8* %self, i8* %r_num, i8* %r_min, i8* %r_max)
  %1 = call i32 @FIX2INT(i8* %r_num)
  %2 = call i32 @FIX2INT(i8* %r_min)
  %3 = call i32 @FIX2INT(i8* %r_max)
  %4 = icmp sgt i32 %1, %3
  br i1 %4, label %5, label %6
; <label>:5
  br label %12
; <label>:6
  %7 = icmp slt i32 %1, %2
  br i1 %7, label %8, label %9
; <label>:8
  br label %10
; <label>:9
  br label %10
; <label>:10
  %11 = phi i8* [ %r_min, %8 ], [ %r_num, %9 ]
  br label %12
; <label>:12
  %13 = phi i8* [ %r_max, %5 ], [ %11, %10 ]
  ret i8* %13
How we implement C extensions
VALUE* bg_pixels = RARRAY_PTR(rb_funcall(self, rb_intern("pixels"), 0));
VALUE* fg_pixels = RARRAY_PTR(rb_funcall(other, rb_intern("pixels"), 0));

long x = 0;
long y = 0;

for( y = 0; y < other_height; y++ ){
    for( x = 0; x < other_width; x++ ){
        bg_index = ( x + offset_x ) + ( y + offset_y ) * self_width;
        bg_pixels[bg_index] = UINT2NUM(
            oily_png_compose_color(
                NUM2UINT( fg_pixels[x+ y * other_width] ),
                NUM2UINT( bg_pixels[bg_index] )
            )
        );
    }
}
Instead of RARRAY_PTR returning a pointer (a number), return a proper Java object.

Operations like getelementptr can return a new Java object that remembers the original object, and what offset to use.

Let SSA names store Java objects as well as numbers.

The load can then use whatever logic we want to actually read a value from the Java object – reuse normal Ruby array logic.
void*

public final class LLVMTruffleObject {
    private final TruffleObject object;
    private final long offset;
}

The C program has no way of knowing it’s not a real char *

Virtualised pointer – really points to a Java object which represents our Ruby string

```c
char *virtualised_string;
virtualised_string[n] = 'x';
```

Intrinsic operations on the pointer are redirected to be method calls

```ruby
virtualised_string.read_at_offset(n)
```
Evaluation
Evaluation is based on earlier work

• We used to have a C interpreter – TruffleC
• We’ve moved on from this, because we want to support more languages
• But we aren’t able to run all the same benchmarks yet
• So we’ve showing results from our old implementation in the mean time
• We’re pretty sure results will be similar, as the compiled code is similar
Native C extensions give an order of magnitude performance boost

Performance on Ruby C Extensions Oily PNG and PSD Native

Existing attempt to mix managed/native are very disappointing

Performance on Ruby C Extensions Oily PNG and PSD Native

Our solution is 3x faster than native!

Performance on Ruby C Extensions Oily PNG and PSD Native

It’s clear that cross-language inlining is a key part of the performance.

Conclusions
Welcome Chris

Account  Sign Out  Help  Country  Communities  I am a...  I want to...  Search

Products  Solutions  Downloads  Store  Support  Training  Partners  About  OTN

Oracle Technology Network > Oracle Labs > Programming Languages and Runtimes > Downloads

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 (v0.15), Development Kit
- GraalVM preview for Linux (v0.15), Runtime Environment
- GraalVM preview for Mac OS X (v0.15), Development Kit
- GraalVM preview for Mac OS X (v0.15), Runtime Environment

- labsjdk-8u92-jvmci-0.20-darwin-amd64.tar.gz
- labsjdk-8u92-jvmci-0.20-linux-amd64.tar.gz
- labsjdk-8u92-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 sources. All these executables are in the bin directory of GraalVM. You might want to set the JAVA_HOME environment variable to point to the GraalVM directory.
Open Source

- [https://github.com/graalvm/graal-core](https://github.com/graalvm/graal-core)
  - Graal compiler
- [https://github.com/graalvm/truffle](https://github.com/graalvm/truffle)
  - Truffle language implementation framework
- [https://github.com/graalvm/fastr](https://github.com/graalvm/fastr)
  - Fast R runtime
- [https://github.com/graalvm/sulong](https://github.com/graalvm/sulong)
  - Dynamic runtime for LLVM bitcode
- [https://github.com/jruby/jruby/wiki/Truffle](https://github.com/jruby/jruby/wiki/Truffle)
  - Fast Ruby runtime
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 Wolczko
Andreas Wöß
Thomas Würthinger

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

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 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