Towards Ruby 2.0: Progress of (VM) Internals

Koichi Sasada
Heroku, Inc

The results of “My Code”
Agenda

• Background
• Finished work - Ruby 2.0 Internal Changes
  – Support Module#prepend
  – Introducing Flonum
  – New set_trace_func
  – Controllable asynchronous interrupts
  – Deep changes
• Remaining work - Ruby 2.0 Internal Features
  – Virtual machine changes
    • Enable “disabled-optimizing” options
    • Optimize “send” instruction
    • Change VM data structures
  – C APIs for “incomplete features”
• Future work – Dreams: After Ruby 2.0
Introduction

• Koichi Sasada
  – Heroku, Inc.
    • Heroku is a cloud application platform - a new way of building and deploying web apps.
    • No longer a professor 😊
  – One of CRuby committer
    • Full-time committer employed by Heroku, Inc.
    • Toward Ruby 2.0 (next release)
    • Matz is my boss.
Background

Brief History of Ruby Interpreter

1993 2/24
Birth of Ruby
(in Matz’ computer)

1995/12
Ruby 0.95
1st release

1996/12
Ruby 1.0

1998/12
Ruby 1.2

1999/12
Ruby 1.4

2000/6
Ruby 1.6

2003/8
Ruby 1.8

2009/1
Ruby 1.9.0

2012/9
イマココ

2000 Book:
Programming Ruby

2004～
Ruby on Rails
Background
Ruby 2.0 Roadmap

2012/Aug
“Big-feature” freeze (was invalidated?)

2012/Sep
イマココ

2012/Oct
Feature freeze

2013/2/24
Ruby 2.0 Release (20th anniversary)

Quoted from “[ruby-core:40301] A rough release schedule for 2.0.0”
Background
Ruby 2.0 Policy

• Compatibility
• Compatibility
• Compatibility
• Usability
• Performance
Finished work
Ruby 2.0 Internal Changes
Finished work
Ruby 2.0 Internal Changes

• Support Module#prepend
• Introducing Flonum
• New backtrace API “caller_locations”
• New set_trace_func related features
• Controllable asynchronous interrupts
• Deep changes
Module#prepend

• Classes can include modules

```
module M; end
class C; include M; end
```

• The method search order is “Original -> Included module”

• Object#extend allow to put module, but it’s only extend an object
Module#prepend

- Module#prepend enables to extend the class which override original class
Module#prepend

• Example
  – Invoke specific procedure before/after the method invocation (“around”)

```ruby
module EachTracer
  def each(*args)
    before_each
    r = super(*args) # call original
    after_each
    r
  end
end

class Array
  prepend EachTracer
  def before_each; p :befor_each; end
  def after_each; p :after_each; end
end

[1, 2, 3].each{|i| p i} #=>
```

```ruby
  :befor_each
  1
  2
  3
  :after_each
```
Ruby 1.9 or before

class C0; end
module M0; end
module M1; end
class C1 < C0
  include M0
end
obj = C1.new
obj.extend M1

class Singleton
  class
    super
  end
end

Class
Singleton
After Ruby 2.0
Module#prepend

```ruby
class C0; end
module M0; end
module M1; end
module M2; end
class C1 < C0
  include M0
  prepend M1
  prepend M2
end
obj = C1.new
```
Introducing Flonum
(only on 64bit CPU)

• Problem: Float objects are not immediate on Ruby 1.9
  – It causes GC overhead problem

• To speedup floating calculation, represent Float object as immediate object
  – Specified range Float objects are represented as immediate object (Flonum) like Fixnum
    • $1.72723e^{-77} < |f| < 1.15792e+77$ (approximately) and +0.0
    • Out of this range and all Floats on 32bit CPU are allocated in heap
  – No more GCs! (in most of case)
  – Flonum and old Float are also Float classes
  – Proposed by [K.Sasada 2008]
  – On 64bit CPU, object representation was changed
Flonum: Float in Heap (1.9 or before)

All of Float object are allocated in heap

Data structure in heap contains IEEE754/double

On 64bit CPU

HEAD
- T_FLOAT
- Float
- etc

VALUE

IEEE754 Double

8B x 6w = 48 byte for Float object
Flonum: Encoding
IEEE754 double floating number

64bit double

s: sign (1bit)
e: exponent (11bit)
m: mantissa (52bit)

\[-1^s 2^{e-1024} m \quad (m = 1 + \sum_{i=0}^{51} \frac{b_i}{2^{52-i}})\]
Flonum: Range

IEEE754 double

b63 | b60-b62 | b60-b52 | b51-b0

Check if e (b52 to b62) is within 768 to 1279, then it can be represent in Flonum.

This check can be done with b60-b62.

(+0.0 (0x00) is special case to detect)
Flonum: Encoding

IEEE754 double

Ruby’s Flonum

Flonum representation bits (2 bits)
#define FLONUM_P(v) ((v&3) == 2)

☆ +0.0 is special case (0x02)
### Flonum:
Object representation on VALUE

<table>
<thead>
<tr>
<th></th>
<th>Non Flonum</th>
<th>Flonum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixnum</td>
<td>...xxxx xxx1</td>
<td>...xxxx xxx1</td>
</tr>
<tr>
<td>Flonum</td>
<td>N/A</td>
<td>...xxxx xx10</td>
</tr>
<tr>
<td>Symbol</td>
<td>...xxxx 0000 1110</td>
<td>...xxxx 0000 1100</td>
</tr>
<tr>
<td>Qfalse</td>
<td>...0000 0000</td>
<td>...0000 0000</td>
</tr>
<tr>
<td>Qnil</td>
<td>...0000 0100</td>
<td>...0000 1000</td>
</tr>
<tr>
<td>Qtrue</td>
<td>...0000 0010</td>
<td>...0001 0100</td>
</tr>
<tr>
<td>Qundefined</td>
<td>...0000 0110</td>
<td>...0011 0100</td>
</tr>
<tr>
<td>Pointer</td>
<td>...xxxx xx00</td>
<td>...xxxx x000</td>
</tr>
</tbody>
</table>
New backtrace API “caller_locations”

• Problem: caller() returns an array of strings
• Introduce “caller_locations”
  – Returns an array of object which has methods such as “lineno”, “path”, etc
  – caller_locations(0)[0].path #=> “foo.rb”
  – caller_locations(0)[0].lineno #=> 23
• Boost creating backtrace speed by internal changes
  → Speedup exception creation
New “set_trace_func”

• Problem: set_trace_func is not flexible and slow

• TracePoint API
  – TracePoint.trace(events) do block end
  – Specify “events” to invoke
  – Add new events (block_enter, etc)

• New C APIs
  – Rewrite all trace_func related code and enable lightweight probes for profilers
  – x2~ faster for “very lightweight” probes
Controllable asynchronous interrupts

• Problem: Can not make safe program with Async-interrupts such as TimeoutError

• **Thread#control_interrupt(hash) do block end**
  – hash contains exception classes as keys and symbols represents 3 states as values
    • immediate: interrupt immediately
    • on_blocking: interrupt only when before blocking
    • never: never interrupt
  – Async interrupts specified by hash are masked in block
Controllable asynchronous interrupts

# example: Guard from TimeoutError
require ‘timeout’
Thread.control_interrupt(TimeoutError => :never) {
  timeout(10){
    # TimeoutError doesn’t occur here
    Thread.control_interrupt(TimeoutError => :on_blocking) {
      # possible to be killed by TimeoutError
      # while blocking operation
    }
    # TimeoutError doesn’t occur here
  }
}
Deep changes

• Remove “finish” frame from control frames
• Unify “lfp” and “dfp” into “ep”
  – Reduce time of creating method frame (control frame) is important than reduce time of accessing local variables
• Call “allocation function” directly
  – Lightweight object allocation
    – miniruby -l./lib -rbenchmark -e 'GC.disable;Benchmark.bm{|x|x.report{10_000_000.times{Object.new}}}’ #=>
Remaining tasks for Ruby 2.0
Remaining works
Ruby 2.0 Internal Features

• Virtual machine changes
  – Enable “disabled-optimizing” options
  – Optimize “send” instruction
  – Change VM data structures

• More supports for Profiler/Debugger

• C APIs for “incomplete features”
Virtual machine changes
Enable “disabled-optimizing” options

• Operand/Instruction unification
  – Unify instruction and specific operand and make additional instructions
  – Macro instructions

• Tail-call optimization?
  – Eliminate method frame if the call is tail-call
  – To avoid stack overflow
  – I prefer auto expanding VM stack than it
Unification result
Virtual machine changes
Optimize “send” instruction

Ruby is OO language
→ Everything are methods
→ Optimizing Method call is important
Virtual machine changes
Optimize “send” instruction

• Inline cache
  – More sophisticate cache invalidation protocol
  – Cache checking results such as visibilities, number of parameters and so on
  – Direct C function call path

• Compilation (future work)
  – JIT compilation only for “send” instruction is feasible / cost effective
  – Need to consider about maintenance
Virtual machine changes
Change VM data structures

• Reconstruct rb_control_frame_t
  – make it small to boost pushing new frames
• Introduce rb_code_context_t to represent code context (iseq + cref + method info + ...)
• Re-arrange boot sequence to make it easy for application embedded usage
class C
CONST = 1
@iv = 2
@@cv = 3
def initialize
  @i = @j = @k = 0
end
def a
  # ...
end
end
alias b a
end

Current method table data structure
Current ISeq data structure

ISeq

static data:
- name
- filename
- arg info
- bytecode body
- catch_table entries
- line info entries
- local_table (ID *)
- stack_max
- parent_iseq (直近の親)
- local_iseq (lfp 指し先)
- mark ary

dynamic data:
- self
- orig
- inline cache entries
- klass (def したクラス)
- cref_stack
- defined_id (define_method されたときのid)

Inline cache entry
key: vmstat
class
value (union):
- value
- method
- index

Line info entry
position (このPCは)
line_info (何行目)

CREF stack entry
visibility
class
class C
class D
def m
1.times{
  # block1
  1.times{
    # block2
  }    
}    
end
end

def m_c
end
end

Bytecode body
Insn 1 Op1 ptr Op2 ptr Insn 2 Insn 3 Op1 ptr Op2 lm Op3 lm ...

mark ary
- String “foo”
- Regexp /bar/
- ISeq obj
The proposed method table data structure is represented in the diagram above. The `RBClass` structure contains the `rclass_ext_t m_tbl`, `iv_index_table`, `iv_index`, and `rb_classext_struct`. The `iv_index` table maps indices to method entries, such as:

- `@i` maps to `initialize`
- `@j` maps to `a`
- `@k` maps to `b` (alias of `a`)

The `rb_classext_struct` includes the `super: Object` and `iv_tbl` and `const_tbl` entries. The `const table` contains `CONST 1` and the `iv table` contains `@iv 2` and `@@cv 3`. The `method table` contains method entries for `initialize`, `a`, and `b` (alias of `a`). Each method entry has fields such as `method_flag`, `klass`, `callee_id`, `original_id`, `alias_count_ptr`, `method_type`, and `cref_stack`.

The code snippet demonstrates the class definition:
```ruby
class C
  CONST = 1
  @iv = 2
  @@cv = 3
  def initialize
    @i = @j = @k = 0
  end
  def a
    # ...
  end
end
alias b a
```

The diagram visually represents these structures and method entries, showing how they are organized in the method table data structure.
### rb_method_info_t
- **method_flag:** public
- **callee_id:** a
- **orig_id:** a
- **klass**
- **alias_ptr**
- **method_other_info:** 2

### rb_code_context_t
- **type:** method | iseq
- **iseq**
- **cref_stack**
- **other_info**

### rb_iseq_t
- **static data:**
  - rb_code_context_t: type iseq cref_stack other_info
  - name
  - filename
  - arg info
  - bytecode body
  - catch_table entries
  - line info entries
  - local_table (ID *)
  - stack_max
  - parent_iseq (直接の親)
  - mark ary
  - self

### Proposed code context (iseq) data structure
- Value stack
- Local[1] code

### Method Table
- **initialize**
- **a**
- **b (alias of a)**

### Diagram Notes
- ISeq RVALUE
- Local[1] code
- Value stack
- Proposed code context (iseq) data structure
def foo
    ...
} end
def bar
    yield
end
foo
def foo
  bar{
  ..
  }
end

def bar
  yield
end

foo
```
def foo
    bar{
        ...
    }
end
def bar(&b)
    b.call
end
foo
```
C APIs for “incomplete features”

• Discussion about several features are not completed
  – “require” framework
  – Instrumentation framework
  – ...

• Strategy: Introduce “C APIs” secretly and experiment new features in gem
Future work

Dreams: After Ruby 2.0

• Compilation
• Parallel execution
• Pluggable features
Student’s research: CastOff
A performance improvement tool for ruby1.9.3

(1) Use from ruby script
- require ‘cast_off’
- CastOff.compile(Klass, :Method, binding, TypeInfo)
- ... 
- Compile Klass#Method
- Load compiled binary
- Run faster

(2) Use from command line
- $ CastOff “program”
- Run and Profile “program”
- Compile methods in “program”
- Run faster

Slides from RubyConf2011
Compilation flow

# Tuning code
class Sample
  CastOff.compile(
    self, :sample, binding, :f => Foo)
end

Ruby program

Code Manager

Configuration

Compilation Target

Code Generator

Makefile

C Source

C build tools

C Extention

Configuration

Slides from RubyConf2011
Performance improvements

Execution time ratio (CRuby 1.9.3 / CastOff)

Guard vs. NoGuard

Slides from RubyConf2011
Parallel Execution

• Run threads in parallel (JRuby, MacRuby, …)
  – Good: Well known approach
  – Bad: Difficult to make safe/correct multi-threaded programs
    • Many tragedy (in Java, etc.)
  – Bad: Difficult to make efficient implementation with fine-grain lock
• Parallel processes (dRuby, …)
  – Good: No need to implement
  – Bad: Marshal overheads
Support friendly
Coarse-grained parallel computing

• Encourage Multi-process
  – Traditional well-known approach
  – Toward advanced dRuby

• Multi-VM
  – VMs in one process
  – Light-weight communication
Student’s research
Tunnel: Inter-process communication w/shared memory

• Object transfer w/ shared memory
Student research (cont.)
Space: Inter-process
Space w/shared memory

• Shared space between ruby processes
  – Similar to Linda/Rinda
Evaluation of Tunnel

Compared with pipe (Marshal)

- true
- false
- nil
- Fixnum
- Float
- Bignum
- Complex (Fixnum)
- Complex (Float)
- Rational (Fixnum)
- Rational (Bignum)
- String (embedded)

Send Object
Bignum: 10^100
Float: 10^-3
String: "a"
Parallel Execution
MVM
Parallel Execution
Multiple-VM (MVM) on Ruby

• Multiple VMs in one process
  – VMs are completely isolated (Each VM has an independent object space (heap))
  – VMs run in parallel
    • Each VM has own GVL (w/o fine grained lock)

Ruby process

<table>
<thead>
<tr>
<th>Ruby VM</th>
<th>GVL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Serial execution w/GVL</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ruby VM</th>
<th>GVL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Channel

- Passing References
- Memory copy

Slides from RubyConf2011
Our Approach
Multiple-VM (MVM) on Ruby

• Channel: Inter-VM Communication mechanism
  – The only way to communicate with other VMs
  – Simply passing references or copying memory in the same address space
Evaluation

HTML rendering app

• Master dispatch string to worker and worker returns rendered HTML.
Evaluation

HTML rendering app

![Graph showing speedup ratio vs. number of VMs/Processes for Processes+pipe and MVM+channel methods. The graph indicates that MVM+channel has a higher speedup ratio compared to Processes+pipe as the number of VMs/Processes increases.]
Evaluation
DB app

- Benchmark assuming web application
- Several front-end VMs and one DB VM
- YUBIN-Number (zip-code) DB on memory
- Using **dRuby** (w/MVM) framework

![Diagram showing Ruby process flow with connections between Front-end VM, Query String (YUBIN#), Result String (Address), and DB VM with On memory YUBIN# DB.]

Slides from RubyConf2011
Evaluation

DB app

Query/sec

DB benchmark

MVM + Channel

fork + TCP

Slides from RubyConf2011
Future work on MVM/Parallel

- Extend this communication channel between inter-process (w/ shared memory), inter-node
- Migratable Ruby activity (threads, blocks (closures) and so on)
Summary

• Finished work - Ruby 2.0 Internal Changes
• Remaining work - Ruby 2.0 Internal Features
• Future work – Dreams: After Ruby 2.0

We will release Ruby 2.0 next year! 
Don’t miss it!
Thank you!

Koichi Sasada
Heroku, Inc.
ko1@heroku.com