Growing the Ruby Interpreter

Koichi Sasada
<ko1@heroku.com>
Today’s talk

• Ruby 2.1 and Ruby 2.2
• How to grow up the Ruby interpreter?
  • Evaluator
  • Threading
  • Object management / Garbage collection
Koichi Sasada as a Japanese

Growing the Ruby interpreter, Koichi Sasada, RubyConf Brasil 2014
Koichi Sasada as a Programmer

• CRuby/MRI committer
  • Core components developer
  • Virtual machine (YARV) since 2004/1/1
  • Rewrote Threads, GC, and so on

• Matz team at Heroku, Inc.

• Director of Ruby Association
• Foundation to encourage Ruby dev. and communities

• Activities
  • Ruby programmer certification program
    • http://www.ruby.or.jp/en/certification/examination/ in English
  • Grant project. Submit your proposal now!
    • 3 projects. About 5,000 USD per project (deadline: 3rd Oct)
  • Ruby Prize
  • Maintenance of Ruby (Cruby) interpreter
  • Events, especially RubyWorld Conference
  • Donation for Ruby developments and communities
Heroku, Inc. http://www.heroku.com

Grow the Ruby interpreter, Koichi Sasada, RubyConf Brasil 2014
Heroku, Inc. http://www.heroku.com
Ask Nando Vieira for more details

Growing the Ruby interpreter, Koichi Sasada, RubyConf Brasil 2014
• Heroku supports OSSs
  • Many talents for Ruby, and also other languages
  • Heroku employs 3 **Ruby interpreter core developers**
    • Matz
    • Nobu
    • Ko1 (me)
• We name our group “Matz team”
Matz
Famous title collector

- He has so many (job) title
  - Chairman - Ruby Association
  - Fellow - NaCl
  - Chief architect, Ruby - Heroku
  - Research institute fellow – Rakuten
  - Chairman – NPO mruby Forum
  - Senior researcher – Kadokawa Ascii Research Lab
  - Visiting professor – Shimane University
  - Honorable citizen (living) – Matsue city
  - Honorable member – Nihon Ruby no Kai
  - ...

- This margin is too narrow to contain
Nobu
Patch monster

• Great patch creator
Nobu is Great Patch Monster

Growing the Ruby interpreter, Koichi Sasada,
RubyConf Brasil 2014
EDD: Event Driven Development

Growing the Ruby interpreter, Koichi Sasada,
RubyConf Brasil 2014
“Mission of Matz team”

Improve quality of next version of CRuby
“Mission of Matz team”

• **Improve quality of next version of CRuby**
  • Matz decides a spec finally
  • Nobu fixed huge number of bugs
  • Ko1 improves the performance

• **Next version of CRuby is “Ruby 2.2.0”**
Ruby 2.1
Current stable
Ruby 2.1

a bit old Ruby

• **Ruby 2.1.0** was released at **2013/12/25**
  • New features
  • Performance improvements

• **Ruby 2.1.1** was released at **2014/02/24**
  • Includes many bug fixes found after 2.1.0 release
  • Introduce a new GC tuning parameter to change generational GC behavior (introduce it later)

• **Ruby 2.1.2** was released at **2014/05/09**
  • Solves critical bugs (OpenSSL and so on)
Ruby 2.1 New syntax

• New syntaxes
  • Required keyword parameter
  • Rational number literal
  • Complex number literal
  • `def` returns symbol of method name

Growing the Ruby interpreter, Koichi Sasada, RubyConf Brasil 2014
Ruby 2.1 Runtime new features

• String#scrub
• Process.clock_gettime
• Binding#local_variable_get/set
• Bignum now uses GMP (if available)
• Extending ObjectSpace
Performance improvements

- Optimize “string literal”.freeze
- Sophisticated inline method cache
- Introducing Generational GC: RGenGC
RGenGC
Performance evaluation (RDoc)

About x15 speedup!

Growing the Ruby interpreter, Koichi Sasada,
RubyConf Brasil 2014
We are here!

2013/12
Ruby 2.1.0

RubyConf Brasil
8/28, 29

RubyKaigi
9/18, 19, 20

RubyConf
11/17, 18, 19

Events are important for
EDD (Event Driven Development) Developers

Growing the Ruby interpreter, Koichi Sasada,
RubyConf Brasil 2014
Ruby 2.2 (rough) schedule

2013/12
Ruby 2.1.0

We are here!

2014/12/25
Ruby 2.2.0

Sep/2014
Preview 1
Big feature freeze

Sep/2014
Dev. Meeting
Feature proposal

Nov/2014
Preview 2
Feature freeze

Dec/2014
Release candidate

Bug fix only

Critical Bug fix only

Growing the Ruby interpreter, Koichi Sasada,
RubyConf Brasil 2014
2.2 big features (planned)

• New syntax: not available now
• New method: no notable methods available now
• Libraries:
  • Minitest and test/unit will be removed (provided by bundled gem)
2.2 internal changes

• Internal
  • C APIs
    • Hide internal structures for Hash, Struct and so on
    • Remove obsolete APIs
  • GC
    • Symbol GC (merged recently)
    • More ages strategy to reduce too-fast-promotion
    • Incremental GC to reduce major GC pause time
• VM
  • More sophisticated method cache
Ruby 2.2 internals
Symbol GC

1_000_000.times{|i| i.to_s.to_sym}
p Symbol.all_symbols.size

# Ruby 2.1
#=> 1,002,376

# Ruby 2.2 (dev)
#=> 25,412
NOTE: Drink a drop of water
Growing up the Ruby Interpreter

How do we grow up the Ruby interpreter?
Software consists of many components
Ruby’s components for users

Ruby (Rails) app

Standing on the shoulders of giants

i gigantum umeris insidentes

So many gems
such as Ruby on Rails (ActiveSupport, ...) and so on.

RubyGems/Bundler

Ruby interpreter

Growing the Ruby interpreter, Koichi Sasada,
RubyConf Brasil 2014
Interpret on RubyVM

Ruby’s components from core developer’s perspective

Ruby script

Parse

Compile

Ruby Bytecode

Bundled Libraries

Gem Libraries

Embedded classes and methods (Array, String, ...)

Threading

Evaluator

Object management (GC)

Interpret on RubyVM

Growing the Ruby interpreter, Koichi Sasada, RubyConf Brasil 2014
My contributions

- Improve the performance for
  - Evaluator (10 years)
  - Thread management (10 years)
  - Memory management (recent years)
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

2013/2
Ruby 2.0

2013/3
RGenGC

2013/12
Ruby 2.1.0

2004/1
YARV development

Growing the Ruby interpreter, Koichi Sasada,
RubyConf Brasil 2014
Grow up Ruby interpreter by modification of core components
Parser

Compile

Ruby Bytecode

Parse

Compile

Evaluator

Threading

Object management(GC)

Embedded classes and methods (Array, String, ...)

Bundled Libraries

Gem Libraries

Interpret on RubyVM

Growing the Ruby interpreter, Koichi Sasada, RubyConf Brasil 2014
Evaluator

- Named YARV: Yet another RubyVM
  - Start until 10 years ago (2004/01/01)
  - Simple stack machine architecture
  - Execute each bytecode instructions one by one
- Apply many known optimization techniques
Evaluator
Compile Ruby to AST

Ruby Program

a = b + c

Abstract Syntax Tree

a = Method Dispatch(:+)

b
c

Growing the Ruby interpreter, Koichi Sasada,
RubyConf Brasil 2014
Evaluator
Compile AST to Bytecode

Abstract Syntax Tree

```
a =
Method Dispatch(:+)
b c
```

Tree data

VM Instructions
```
getlocal b
getlocal c
send +
setlocal a
```

Sequential data

Growing the Ruby interpreter, Koichi Sasada, RubyConf Brasil 2014
Evaluator
Execution as stack machine

Ruby Program
a = b + c

YARV Instructions
- getlocal b
- getlocal c
- send +
- setlocal a

b+c
b
b+c

VM Stack

Compile

Growing the Ruby interpreter, Koichi Sasada,
RubyConf Brasil 2014
Evaluator
Optimizations

• Apply many techniques to improve performance
  • Peephole optimizations
  • Specialized instructions
  • Stack frame layout
  • Efficient exception handling
  • Efficient block representation
  • Direct threading
  • Stack caching
  • Instructions and operands unifications
  • ...

Growing the Ruby interpreter, Koichi Sasada,
RubyConf Brasil 2014
Evaluator
Optimizations: Basic concept

• Analysis usage
  And optimize for frequent cases

• Example: Exception handling
  • Exceptions occur *EXCEPTIONAL* so optimize for no-exception control flow
Performance evaluation compare with Ruby 1.8

Higher is good
Main components

• Evaluator
• Thread management
• Memory management
Interpret on RubyVM

- Ruby script
- Parse
- Compile
- Ruby Bytecode
- Threading
- Evaluator
- Object management (GC)
- Bundled Libraries
- Gem Libraries
- Embedded classes and methods (Array, String, ...)

Growing the Ruby interpreter, Koichi Sasada,
RubyConf Brasil 2014
Threading

• Using native threads for each Ruby threads
• Parallel ruby execution is prohibited by GVL
  • You can free GVL if you write a code carefully in C level and run it in parallel
Threading
Ruby 1.8 and before

One OS (native) thread manages all Ruby threads
This technique is a.k.a. Green Thread
Threading
Layered view

Growing the Ruby interpreter, Koichi Sasada,
RubyConf Brasil 2014
Threading
Ruby 1.9 and later

Native threads with Giant VM Lock

Growing the Ruby interpreter, Koichi Sasada,
RubyConf Brasil 2014
Threading
Layered view

Growing the Ruby interpreter, Koichi Sasada, RubyConf Brasil 2014
Threading

Why not green threads?

• Advantage of green threads
  • Lightweight creation

• Disadvantage of green threads
  • Slow context switching (under portable way)
  • Need to take care for blocking methods
    • Such as network read/write
  • Difficult to collaborate with other C libraries using threads

We don’t need to make threads frequently. (and we also have Fiber)
Threading
Why GVL?

• To protect Ruby users from nightmare debugging
  • Shared parallel threading can make non-deterministic bugs which is too hard to debug
  • “Thread programming is too difficult for human being”

• Disadvantage
  • CRITICAL ISSUE: No parallel programming in Ruby
  • Need another programming model for parallel
    • Current *SHARED EVERYTHING* model is not match
    • Correct isolation level for each parallel execution units
Threading
How to make parallel ruby program?

• Now:
  • Use parallel threads provided by JRuby/Rubinius
    • If you think you can make correct thread programs
  • Use process (for example, w/ parallel gem, w/ dRuby)

• Future:
  • Introduce smart conventions to avoid threading bugs
    • Matz likes Actor model (Erlang)
  • Introduce limited shared memory model
  • Introduce smart debugging feature
    • Detecting bugs, avoid nondeterministic behaviors, ...
Interpret on RubyVM

Object management (GC)

- Ruby script
- Parse
- Compile
- Ruby Bytecode
- Threading
- Evaluator
- Embedded classes and methods (Array, String, ...)
- Object management (GC)
- Bundled Libraries
- Gem Libraries

Growing the Ruby interpreter, Koichi Sasada, RubyConf Brasil 2014
Object and memory management

• “Object.new” allocate a new object
  • “foo” (string literal) also allocate a new object
  • Everything are objects in Ruby!
• We don’t need to “de-allocate” objects manually
Garbage collection
The automatic memory management

Fig. 109. — A Garbage Collector.
http://www.flickr.com/photos/circasassy/6817999189/

Growing the Ruby interpreter, Koichi Sasada,
RubyConf Brasil 2014
Automatic memory management

Basic concept

• Garbage collector recycled “unused” objects automatically
Mark & Sweep algorithm

1. Mark reachable objects from root objects

2. Sweep unmarked objects (collection and de-allocation)

Collect unreachable objects

Growing the Ruby interpreter, Koichi Sasada, RubyConf Brasil 2014
RGenGC: Restricted Generational GC
Generational GC (GenGC) from Ruby 2.1

• Weak generational hypothesis:
  “Most objects die young”

→ Concentrate reclamation effort only on the young objects

Growing the Ruby interpreter, Koichi Sasada, RubyConf Brasil 2014
Generational hypothesis

Object lifetime in RDoc
(How many GCs surviving?)

95% of objects dead by the first GC
Generational GC (GenGC)

- Separate young generation and old generation
  - Create objects as young generation
  - Promote to old generation after surviving $n$-th GC
  - In CRuby, $n == 1$ (after 1 GC, objects become old)
    - $n == 2$ or 3 from Ruby 2.2

- Usually, GC on young space (minor GC)
- GC on both spaces if no memory (major/full GC)
GenGC [Minor M&S GC] (1/2)

- Mark reachable objects from root objects.
  - Mark and **promote to old generation**
  - Stop traversing after old objects

→ **Reduce mark overhead**

- Sweep not (marked or old) objects

- Can’t collect Some unreachable objects

Don’t collect old object even if it is unreachable

1st MinorGC

Root objects
GenGC [Minor M&S GC] (2/2)

- Mark reachable objects from root objects.
  - Mark and **promote to old generation**
  - Stop traversing after old objects

→ **Reduce mark overhead**

- Sweep not (marked or old) objects

- Can’t collect Some unreachable objects

Don’t collect old object even if it is unreachable.
GenGC [Major M&S GC]

- Normal M&S
- Mark reachable objects from root objects
  - Mark and **promote to old gen**
- Sweep unmarked objects
  - **Sweep all unreachable (unused) objects**
RGenGC
Performance evaluation (RDoc)

About x15 speedup!

Growing the Ruby interpreter, Koichi Sasada,
RubyConf Brasil 2014
RGenGC
Performance evaluation (RDoc)

* 12% improvements compare with w/ and w/o RGenGC
* Disabled lazy sweep to measure correctly.
RincGC: Restricted incremental GC
RincGC
Background and motivation

• Ruby 2.1 had introduced generational GC
  • Short marking time on minor GC
  • Improve application throughput

• Still long pause time on major GC
  • Long pause time affects user response time
Proposal:
RincGC: Incremental GC for major GC

• Introducing incremental GC to reduce pause time
• Can combine with Generational GC

<table>
<thead>
<tr>
<th></th>
<th>Generational GC</th>
<th>Incremental GC</th>
<th>Gen+Inc GC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput</td>
<td>High</td>
<td>Low (a bit slow)</td>
<td>High</td>
</tr>
<tr>
<td>Pause time</td>
<td>Long</td>
<td>Short</td>
<td>Short</td>
</tr>
</tbody>
</table>

Growing the Ruby interpreter, Koichi Sasada, RubyConf Brasil 2014
RincGC: Base idea
Incremental GC algorithm

• Move forward GC processes incrementally
  • Mark slots incrementally
  • Sweep slots incrementally

• Incremental marking in 3 phase
  • (1) Mark roots (pause)
  • (2) Mark objects reachable from roots (incremental)
  • (3) Mark roots again, and mark remembered objects (pause)

• Mark objects with three state (white/grey/black)
  • White: Untouched objects
  • Grey: Marked, and prepare to mark directly reachable objects
  • Black: Marked, and all directly reachable objects are marked

• Use write barriers to avoid marking miss from marked objects to live objects
  • Detect new reference from black objects to white objects
  • Remember such source black objects (marked at above (3))

Growing the Ruby Interpreter, Koichi Sasada,
RubyConf Brasil 2014
RincGC:
Incremental GC for CRuby/MRI

• Incremental marking
  • (1) mark roots (gc_mark_roots())
  • (2) Do incremental mark at rb_newobj_of()
  • (3) Make sure write barrier with WB-protected objects
  • (4) Take care of **WB-unprotected objects** (MRI specific)

• Incremental sweeping
  • Modify current lazy sweep implementation
RincGC: Incremental marking

• (1) mark roots (gc_mark_roots())
  • Push all root objects onto “mark_stack”

• (2) Do incremental mark at rb_newobj_of()
  • Fall back incremental marking process periodically
  • Consume (pop) some objects from “mark_stack” and make forward incremental marking

• (3) Make sure write barrier with WB-protected objects
  • Mark and push pointed object onto “mark_stack”

• (4) Take care of **WB-unprotected objects** (MRI specific)
  • After incremental marking (“mark_stack” is empty), re-scan all roots and all living non-WB-protected objects
  • WB-unprotected objects are represented by bitmap (WB_UNPROTECTED_BITS)
RincGC: Diagram

Growing the Ruby interpreter, Koichi Sasada, RubyConf Brasil 2014
Growing up the Ruby Interpreter

How do we grow up the Ruby interpreter?
Growing the Ruby interpreter, Koichi Sasada, RubyConf Brasil 2014

**DO EVERYTHING! NO SILVER BULLET!**

- **Ruby script**
- **Parse**
- **Compile**
  - **Ruby Bytecode**
  - **Threaded classes and methods** (Array, String, ...)
  - **Evaluator**
  - **Object management (GC)**

- **Interpret on RubyVM**
- **Bundled Libraries**
- **Gem Libraries**
DO EVERYTHING!
NO SILVER BULLET!

Loop do
  • Survey techniques
  • Implement techniques
  • Invent new techniques
  • Evaluate techniques
end # endless

Growing the Ruby interpreter, Koichi Sasada,
RubyConf Brasil 2014
DO EVERYTHING!
NO SILVER BULLET!

We did.
We are doing.
We will do!!

Only continuous effort improves software quality.

Growing the Ruby interpreter, Koichi Sasada,
RubyConf Brasil 2014
Future work: Many many many!!

• Evaluator
  • JIT compilation
  • More drastic optimizations

• Threading
  • Parallel execution model (not a threading?)

• Object management and GC
  • Compaction GC
  • Lightweight object allocation
  • CoW friendly memory management

• And more
Summary

• Ruby 2.1 and Ruby 2.2
• How to grow up the Ruby interpreter?
  • Evaluator
  • Threading
• Object management / Garbage collection
Summary

• Ruby 2.1 and Ruby 2.2
• How to grow up the Ruby interpreter?

My answers is:

#=> Continue software development (with love?)
Thank you for your attention

Koichi Sasada
<ko1@heroku.com>