Implementation of Ruby 1.9.3 and later

Koichi Sasada
Department of Creative Informatics,
Graduate School of Information Science and Technology,
The University of Tokyo
Today’s topic

- Ruby 1.9.3
- Next and next version of MRI
We released!

Ruby 1.9.3.3...rc1
‘‘I will release Ruby 1.9.3 within this two weeks unless any serious problem is reported. If you have any trouble with Ruby 1.9.3, please let us know. ’’

It’s means…
We need human sacrifice

http://www.flickr.com/photos/babomike/4741964697/
Ruby 1.9.3
New Features

RTFN

Read the F*****g News
Ruby 1.9.3 Features

- License
  - Original and (GPLv2 → 2-clause BSDL)
- Syntax, Methods/Libraries
  - Private constants
  - DL/YAML → Fiddle/Psyck
  - Test::Unit → Parallel extension (by @sora_h, talking at next room)
  - String#prepend, IO.write, etc
  - … (Actually, I don’t know about Ruby world)
Implementation of Ruby 1.9.3 Internals and later

Koichi Sasada
Department of Creative Informatics, Graduate School of Information Science and Technology, The University of Tokyo
Ruby 1.9.3

Interpreter internals

• Garbage Collection
  • Lazy Sweep
  • Parameter tuning

• Timer thread implementation
  • Reducing power consumption

• GVL implementation for multi-core
  • Solve an problem about non-thread switching on multi-core
Garbage Collection

http://www.flickr.com/photos/ninithedreamer/4649075746/
Introducing Lazy Sweep GC

**Before 1.9.3:** Stop the world mark and sweep

**After 1.9.3:** Stop the world mark, and incremental sweep (shorter stopping time)
✓ You can tune GC by using environment variables.
✓ RUBY_GC_MALLOC_LIMIT
✓ RUBY_HEAP_MIN_SLOTS
✓ RUBY_FREE_MIN

Read a source code for details :p
Quoted from nari3’s slide
Ruby 1.9.3
All About Timer Thread (core, details)

Koichi Sasada
Department of Creative Informatics, Graduate School of Information Science and Technology, The University of Tokyo
Power consumption

http://www.flickr.com/photos/livingos/2494500294/
Background
Timer Thread?

- Timer thread is an internal mechanism on Ruby 1.9
  - For thread scheduling
  - For signal delivery
Problem
Power Consumption?

- Timer thread awake every 10ms
- To notify a running thread to release GVL
- Wake-up anytime even if it is not necessary (no scheduling is needed) → CPU can’t sleep enough
- A few people claimed that the timer thread consumes power

![Diagram showing timer thread and Ruby thread behavior](image-url)
Solution
Rewrite Timer Thread Code

- Make timer thread more smart
  - Timer thread **sleeps infinitely** if only one thread is needed
  - Using **pipe trick**
Evaluation (?)

Power consumption

- 1.9.2
- proposed
Evaluation

Reduce 0.3W on 1 Ruby process
Caution! Incompatibility

- Pipe trick needs additional file descriptors
  - You should not close them
- **Passenger does it!!**
- Skip to close such file descriptors w/ new C API "rb_reserved_fd_p()"
Thread Scheduler on the multi-core

http://www.flickr.com/photos/marksze/4231114748/
Ruby 1.9.3
GVLおよびロックの改善
GVL and Lock improvement

He is a Ruby and Linux kernel committer

Fujitsu Motohiro Kosaki
Background again
Only one thread can run w/GVL

- Timer thread is an internal mechanism on Ruby 1.9
  - For thread scheduling
  - For signal delivery
doesn’t work some program on Multi-core environment

# example
f = false
Thread.new { 
  ....;
  f = true
}
Thread.pass until f
Why?
Good old OS scheduler

Quoted from Kosaki-san’s slide
New fast OS scheduler

Same number of CPU cores

Quoted from Kosaki-san's slide
Situation (1)

Yield!

Return into Queue

( ^ω^ )

I can expect resume!

Quoted from Kosaki-san’s slide
Situation (2)

Resume again!

I can’t resume again…

Quoted from Kosaki-san’s slide
### Solution

- We change GVL passing strategy
  - Passing GVL definitely
  - Ask me later if you want to know

```c
// Quoted from Kosaki-san's slide

gvl_acquire
mutex_lock(&lock->lock)
cond_wait(&lock->wait)
lock->acquired = 1;

// notice
cond_signal(&lock->switch_cond)
mutex_unlock(&lock->lock)

gvl_release
mutex_lock(&lock->lock)
lock->acquired = 0;
cond_signal(&lock->wait_cond)

// sleeping until
cond_wait(&lock->switch_cond)
mutex_unlock(&lock->lock)
```
Thread related performance

Quoted from Kosaki-san's slide
Future

http://www.flickr.com/photos/viernullvier/5698630227/
Implementation of Ruby 1.9.3 and later

Koichi Sasada
Department of Creative Informatics,
Graduate School of Information Science and Technology,
The University of Tokyo
Now discussing which version should be released

- Matz says “Next version is 2.0 including several new syntax”
  - keyword argument support for method definitions
  - Module#mix
  - Module#prepend
  - and others (refinement, classbox, or method shelter?)
- Another says “Should be 1.9.4”
Ruby 2.0?
What feature do you want?

Teach me here, raise your hand! Anything is okay.
Okay.

There are many requirements.
Yesterday, I asked Matz what we should do. He doesn’t want to release 1.9.4 anymore. He wants to say there are no progress in 1.9 series any more, but only in 2.0 series. If spec is concluded, he wants to shrink 2.0 spec and release force.

Please ask Matz tomorrow, if you have opinion about it.
My concerns on next version

- Performance
  - This is why I’m here
- Profiler/debugger interface
  - Please teach me what information you want to know!
- Compiling supports
- I’ll introduce several possibility and activities about “Ruby in Future”
• We add more support for profiling/debugging
• Please teach me what information you want to know!
2 days ago, I posted a patch of `ObjectSpace.reference_from(obj)`

I’m sad there are few feedback…

```ruby
ObjectSpace.reference_from(obj)
#=>
{98752943875 => x, 48334232r23 => y}
# keys are object id
# value are object themselves
```
Research of my student: Performance profiler

- **ll-prof**
  - Real-time profiler for LL languages
    - Ruby and Python (he is Pythonian)
  - Viewer in JavaScript
    - You can see results in your browser
Demo
(movie made by student)
Performance

- Speed
  - VM w/ runtime performance
  - GC performance
- Memory consumption
- Power consumption
Performance
VM w/ runtime speed

- Performance improvements VM
- Easy escape analysis to prevent generating temporary objects
Performance VM w/ runtime speed

- Semi-automatic Type inference and translate Ruby code into C (C extension)
Student’s research: CastOff
A performance improvement tool for Ruby 1.9.3

(1) Use from ruby script

Programmer

require ‘cast_off’
CastOff.compile(Klass, :Method, binding, TypeInfo)
...

• Compile Klass#Method
• Load compiled binary
• Run faster

(2) Use from command line

Programmer

$ CastOff “program”

• Run and Profile “program”
• Compile methods in “program”
• Run faster

Quoted from Shiba-san’s slide
# Tuning code

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

Quoted from Shiba-san’s slide
Performance improvements

Execution time ratio (CRuby 1.9.3 / CastOff)

Faster

Guard  NoGuard

bm_nested_loop.rb  bm_mandelbrot.rb  bm_tera.rb

bm_sieve.rb  bm_lists.rb  bm_rdoc

Quoted from Shiba-san’s slide
Performance
Garbage collection

• MRI/CRuby’s GC is slow
  • Stop the world and mark all
• Generational?
  • Issue: C extension compatibility
• Ideas
  • Parallel GC → Yesterday’s nari3’s talk 😊
  • Escape analysis and reduce GC managed objects
  • Smart data-structure to reduce GC managed objects
Performance
Memory consumption

- One Rails app consume 100MB
  → Encourage sharing resources
    - such as Strings, Arrays, and so on
    - REE/Kiji CoW friendly, generational
Performance
Power consumption

- Time thread modification at 1.9.3 → small one
- More aggressive way?
Performance
Parallel computing

http://www.flickr.com/photos/juror8/394285511/
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

Quoted from Nkagawa-san’s slide
Student research (cont.)
Space: Inter-process
Space w/shared memory

- Shared space between ruby processes
- Similar to Linda/Rinda

![Diagram of shared space between Ruby processes with shared memory.]

Quoted from Nkagawa-san’s slide
Evaluation of Tunnel

Compared with pipe (Marshal)

Send Object
- Bignum: $10^{100}$
- Float: $10^{-3}$
- String: “a”

Quoted from Nkagawa-san’s slide
A.k.a. vaperware 😞
We have progress on it, this year.
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>Channel</th>
<th>Ruby VM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread</td>
<td>GV L</td>
<td>GV L</td>
</tr>
<tr>
<td>Serial execution w/GVL</td>
<td>- Passing References</td>
<td>- Memory copy</td>
</tr>
<tr>
<td>Thread</td>
<td>GV L</td>
<td>GV L</td>
</tr>
</tbody>
</table>

Diagram:
- 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)
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

Ruby process

- Share the same address space

Ruby VM

- Thread: Serial execution w/GVL

- Channel
  - Passing References
  - Memory copy
Evaluation

HTML rendering app

- Master dispatch string to worker and worker returns rendered HTML.
Evaluation
HTML rendering app

![Graph showing speedup ratio vs. # of VMs/Processes for Processes+pipe and MVM+channel methods.]

- **Processes+pipe**
- **MVM+channel**
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, Front-end VM, Query String (YUBIN#), Result String (Address), DB VM, On memory YUBIN# DB connections.](#)
Evaluation
DB app

Query/sec

DB benchmark

- MVM + Channel
- fork + TCP

Query/sec vs # of Processes / VMs

Query/sec range: 5000 to 0

# of Processes / VMs range: 0 to 25
Future work

- Extend this communication channel between inter-process (w/ shared memory), inter-node
- Migratable Ruby activity (threads, blocks (closures) and so on)
Ruby 1.9.3 will be released soon!

- Be Sacrifice!
- NEWS file and blog posts are good to know changes
- This talk introduces background on Ruby 1.9.3

We are continuing the hacking to the future

Stay tuned to the next announcement
Implementation of Ruby 1.9.3 and later

ささだこういち
ko1@atdot.net, ko1@rvm.jp