YARV
Past, Present and Future

Koichi “ko1” Sasada
ささだこういち
ko1 at atdot dot dot net
RUBY MEETS VM
Merging YARV is not a goal, but a start

YARV: Yet Another RubyVM
Notice

- I can’t speak English well, so I write down all things what I want to say.
  - Do you get ready for opera glasses?
  - Unfortunately, some slides are written in Japanese
- You can ask questions with Japanese, C, Ruby, …, or slow/short English.
- “How to impl. Ruby”, not “How to use Ruby”
- “x50” is too big mouth
  - Maybe x20
Self Introduction

- Koichi (given name) Sasada (family name)
  - ささだ (family name) こういち (given name)
  - 笹⽥ (family name) 耕⼀ (given name)
- Lecturer @ University of Tokyo (Feb. 2008-)
- Only VM developer
  - Don’t have compatibility with Matz
  - Please call me “ko-i-chi”
Agenda

- History of YARV
- Advanced VM Topics
  - Performance
    - Parallel Thread Execution
    - Embedding Float Value
    - JIT Compiler
    - Pre-Compiler
      - “Ruby to Compiled file” Compiler
      - “Ruby to C” Compiler
  - New Feature
    - Multi-VM Creation
    - Customizable Ruby Core
    - Debug/Profile support feature
- Summary
History of YARV

4 Years

1, Jan 2004  Project Start
2004-2005 VM Core, Optimization
  - Supported by MITO youth Project (IPA)
2005-2006 Thread, etc
  - Supported by MITO Project (IPA)
  - 1, Apr 2006 Got a Job (Assistant on U-Tokyo)
2006-2007 etc, etc
  - Supported by MITO Project (IPA)
  - 25, Dec 2007 Got a Ph.D
25, Dec 2007 (GMT) 1.9 Release
FYI

- Ruby 2.0 – since 2003 3/31
- Perl 6 – since 2003 4/1
YARV Policy

- Performance
  - Speed, Speed, Speed
  - Applied many many many many optimization Tech.

- Compatibility
  - C extension API
  - Not language compatibility

- Auto-generation
  - VM description to Concrete VM source code
VM Generator

VM Instruction Description → AOT Compiler → C Source

AOT Compiler → Dis-assembler
AOT Compiler → Assembler
AOT Compiler → Documents

VM Instruction Description → Virtual Machine

Virtual Machine → Compiler (Optimizer)

Compiler (Optimizer) → Future work

Compiler (Optimizer) → Verifier
Enemies of YARV

- Ruby Specification ÷ Matz
  - Ruby Spec kills many optimization techs
  - We love “Dynamic” “Meta” Programming, but…
- Changing Spec is also Nightmare
- Portability
  - We can’t use system depending techs.
- Rivals (not Enemy)
  - Jruby, Rubinius, IronRuby, …
- Peggy work on my Job
Evaluation: Improve case

Max: $x_{20}$
Evaluation: Macro-Benchmark

![Bar chart showing speedup ratios for tDiary, Scheme, XML Diff, and Mandelbrot with different configurations: Base, DTC, DTC+Si, DTC+Si+OU, DTC+Si+OU+IU, DTC+Si+OU+IU+IMC, and DTC+Si+OU+IU+IMC+SC.](image)
Evaluation:

Compare with Other Languages

Faster than Other VMs

Limited by Bignum Calculation
Evaluation: VM doesn’t affect Overhead is not in VM
Advanced VM Topics

- Performance
  - Parallel Thread Execution
  - Embedding Float Value
  - JIT Compiler
  - Pre-Compiler
    - “Ruby to Compiled file” Compiler
    - “Ruby to C” Compiler

- New Feature
  - Multi-VM Creation
  - Customizable Ruby Core
  - Debug/Profile support feature
Ph.D Thesis

Efficient Implementation of Ruby Virtual Machine

Implement a High-Speed Ruby Interpreter

Introduce VM
↓
YARV
(Merged into Ruby 1.9)

Parallel Ruby Execution
↓
Parallel Thread Execution

Ruby Meets VM, Koichi Sasada, EURUKO 2008
Parallel Thread Execution

- Using Native Thread
- Get rid of Giant VM Lock
Method (1)
Ruby Thread and Native Thread (1:N) a.k.a. -1.8 Ruby model

Ruby (YARV)

Native Thread
System S/W

Thread Scheduler

S/W

H/W

Processor(s)

PE: Processor Element, UL: User Level, KL: Kernel Level
Method (2)
Ruby Thread and Native Thread (1:1)

Ruby (YARV)

Native Thread
System S/W

Thread Scheduler

Processor(s)

PE: Processor Element, UL: User Level, KL: Kernel Level
Method (3)
Ruby Thread and Native Thread (N:M)

Ruby (YARV)

Native Thread System S/W

Thread Scheduler

Processor(s)

PE: Processor Element, UL: User Level, KL: Kernel Level
## Discussion

**Ruby Thread and Native Thread**

- Mapping with Native Thread and Ruby Thread

<table>
<thead>
<tr>
<th>Model</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:N</td>
<td>Lightweight Thread Control</td>
<td>Can’t run in Parallel</td>
</tr>
<tr>
<td>1:1</td>
<td>Run in Parallel</td>
<td>Heavyweight Thread Control (Creation, etc)</td>
</tr>
<tr>
<td>N:M</td>
<td>Lightweight Thread Control, Run in Parallel</td>
<td>Complication, Non-Portable</td>
</tr>
</tbody>
</table>

- Accept 1:1 model to make Ruby Simple
  - Depend Performance on Native Thread Libraries
Accepted Method:
Ruby Thread and Native Thread (1:1) ← Ruby 1.9/YARV

PE: Processor Element, UL: User Level, KL: Kernel Level
Introduction of Mutual Exclusion

- Needed at
  1. Global VM Management Data
  2. Object Management / GC
  3. Inline Cache
  4. Thread Unsafe “C” methods
(1) Global VM Management Data

- Managed by Table
  - Variable Name → Value
  - Method Name → Method Body
  - ...

- Introduce Synchronization at Table Operation
  - Get/Set
  - Easy
(2) Object Management/GC

- Synchronous GC

Diagram:

- RT1 (GC Thread)
  - Stop (rt#=4, wt#=0)

- RT2
  - Stop (wt#=1)

- RT3
  - Stop (wt#=2)

- RT4
  - Stop (wt#=3)

- GC start (rt# - wt# = 1)
- Restart
- Wake up

- Restart
- Wake up
- Restart
- Restart

Note:
- rt#: Ruby Threads Number
- wt#: Waiting Threads Number
- Time
(2) Object Management
Lock-Free Object Allocation with Thread Local Free List
(2) Object Management
Lock-Free Object Allocation with Thread Local Free List
(3) Inline Cache

- Using by VM performance
  - Embed Cache Entries in Instruction Sequence
- Sync. For Coherence -> Performance Problem
  - Key and Value of Cache Entry
- Sync.-Free Inline Cache
  - Cache Miss -> Make a new entry
    - GC will clean-up old cache entries
    - Increase Miss-Penalty, but Good for average
(4) Thread Unsafe “C” Methods

- CRuby has many many methods implemented by “C”
  - All of them are “Thread Unsafe”
  - Because -1.8 doesn’t support parallelization

- Basic Policy: Using Giant-Lock
  - Invoke old “C” method with Giant-Lock Acquire
  - Re-write C methods as Thread Safe, this method will be Giant-Lock free
Problem
Conflict of Giant-Lock Acquirement

- GL Conflict → Performance Decrement
- Limit Running CPU
  - Check GL Conflict Periodically
  - Limit their CPU
  - Using `pthread_setaffinity_np` on NTPL
    - `SetThreadAffinityMask` on Windows
Running CPU Limitation

Ruby (YARV)

Native Thread
System S/W

Thread Scheduler

Processor(s)

PE: Processor Element, UL: User Level, KL: Kernel Level
Running CPU Limitation

Ruby (YARV)

Native Thread System S/W

S/W

H/W

Processor(s)

Thread Scheduler

RT1

RT2

⋯

RTn

NT

NT

⋯

NT

GL Conflict

Force to run RT1 and RT2 in Serial → Avoid Conflict

PE: Processor Element, UL: User Level, KL: Kernel Level
Performance Evaluation

Environment

- **Evaluation Environment**
  - CPU: Intel Xeon CPU E5335 2.0GHz
    - Quad core x 2 = 8 core
  - OS: GNU/Linux 2.6.18 x86_64 SMP / NPTL
  - Compiler: gcc version 4.1.2

- **Ruby**
  - *ruby* 1.8.6 (2007-11-02) [x86_64-linux]

- **YARV Optimization**
  - All except Unification, Stack caching
Evaluation

Thread control Primitives

- Creation, Switch: 0.1M, Mutual Exclusion: 1M
- Low Performance for Creation/Join because of Native Thread
  - Native Thread Overhead
  - Memory Allocation Overhead
- High Performance Synchronization
- High Performance Thread Context Switch
  - Independent Stack-Depth (1.8 depends on depth)

<table>
<thead>
<tr>
<th></th>
<th>Ruby (sec)</th>
<th>YARV (sec)</th>
<th>Ratio</th>
<th>NTPL (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creation</td>
<td>0.89</td>
<td>1.95</td>
<td>0.46</td>
<td>0.59</td>
</tr>
<tr>
<td>Mutual Exclusion</td>
<td>0.67</td>
<td>0.38</td>
<td>1.76</td>
<td>-</td>
</tr>
<tr>
<td>Switch (depth:1)</td>
<td>6.01</td>
<td>0.06</td>
<td>100.17</td>
<td>-</td>
</tr>
<tr>
<td>Switch (depth:16)</td>
<td>11.55</td>
<td>0.06</td>
<td>192.5</td>
<td>-</td>
</tr>
</tbody>
</table>
Evaluation
Result (Micro-benchmark)

- **fib**: fib(N) (Make new Thread if N>30)
- **hetero**: fib + concat (1 thread)
- **mandel**: Mandelbrot (Big GC overhead)
- **concat**: String Concatenate (No Parallelism)
Evaluation

Result (Micro-benchmark)

x 7.4 w/8 Cores
CPU Limitation

Using Limitation

No Limitation

![Graph with Speedup Ratio (P=1)](image)

- `concat (w/CPU limit)`
- `concat (wo/CPU Limit)`

x3.4
Parallel Thread Execution

Problem

- Unsafe “Old C” methods
  - Replacing all is not easy task
  - Man Power Problem?

- Programming Model
  - Is Parallel Thread Application easy to write?

- Ruby 1.9
  - 1.9 support Native Thread
  - 1.9 doesn’t support Parallel Thread Execution
Embedded Float Representation

- Float object is not immediate type
  - This means that Float is “allocated” each time
  - Ex) Fixnum, Symbol, etc
- Half execution time of Float calculation is Memory management
Toy-Program

```
i = 0; f = 0.0
while i<30_000_000
    i += 1
    f += 0.1; f -= 0.1
    f += 0.1; f -= 0.1
    f += 0.1; f -= 0.1
    f += 0.1; f -= 0.1
end
```
List of Execution Time
Toy-Program

Using OProfile
Linux 2.6 (x86_64), Xeon
Embed Float Object as Fixnum

- Solution: Embed 64bit Float value to 64bit CPU Pointer type!
Review

IEEE 754 Double Precision Representation

64bit Double

s: Sign (1 bit)

e: exponential (11bit)  m: Mantissa (52bit)

\[-1^s 2^{e-1024} m \quad (m = 1 + \sum_{i=0}^{51} \frac{b_i}{2^{52-i}})\]
Discussion
How to Embed 64 bit Double?

- VALUE embed Object doesn’t need memory overhead
- 64bit CPU have 64 bit pointer type
  → Use 64 bit CPU
- At least we need 1 bit for TAG bit
  - From Mantissa?
    - Decrease Precision
  - From Exponential?
    - Decrease Representation Range
Proposal

- From Exponential
  - But Store Float in Heap if it is Out of Range
    → **Save a Range and Precision**
      - Often used $01000000000b\sim10111111111b$ ($2^{-512}\sim2^{511}$)

- If Float is out of range, alloc from Heap
- **Float Out of this Range is Rare Number on Numeric Application** -> Practical Solution
Proposal
Real Program

- 1) Check the Range of “e” (512〜1535)
- 2) IEEE745 double -> Float (Encoding)
- 3) Float -> IEEE745 double (Decoding)
Proposal

Float Representation with Tag

- $e: 100000000000b \sim 01111111111b$
- Note that if “$b62 \neq b61$”, we can embed.
- On Ruby, Tag is at LSB
  → Left Rotate 3bit
- $b63 \ b62 \ b61 \ b60 \ \cdots \ b0 \rightarrow 3 \text{ bit rotate}$
- $b60 \ \cdots \ b0 \ b63 \ b62 \ b61$
Proposal
Real Encoding Code

```c
VALUE rb_float_new(double dbl) {
    VALUE v1 = BIT_DOUBLE2VALUE(dbl);
    VALUE v2 = BIT_ROTL(v1, 3);
    if (((v2 + 1) & 0x02) // check lower 2 bits
        return v2 | 0x01; // Embed tag
    else {
        if (dbl == 0) // 0.0
            return ruby_float_zero;
        else // alloc from Heap
            return rb_float_new_in_heap(dbl);
    }
}
```
double RFLOAT_VALUE(VALUE v) {
    if (v & 1) {
        VALUE v1 = v ^ ((v >> 1) & 1);
        VALUE v2 = BIT_ROTR(v1, 3);
        return BIT_VALUE2DOUBLE(v2);
    }
    else
        return RFLOAT(v)->float_value;
}
Implementation

- Ruby 1.9.0-0
- Easy to Implementation
- No Spec Changes
Evaluation
Toy-Program

- Reduce Mem Time
- Encode/Decode don’t affect to Performance
Evaluation

Compared with other Ruby Impl.

From Comp. Lang. Shootout [4]
Evaluation

Compared with Other Languages

- Evaluate with other languages
- Note that C/Java use “volatile” to avoid optimization

```plaintext
i = 0; f = 0.0
while i<30_000_000
  i += 1
  f += 0.1; f -= 0.1
  f += 0.1; f -= 0.1
  f += 0.1; f -= 0.1
  f += 0.1; f -= 0.1
end
```
Evaluation
Compared with Other Languages
I’m re-designing to reduce VM instructions to impl. it easy

- Current VM has about 50 instructions
- Ex) “definemethod” move to “Method”
[PLAN]

Pre-Compiler

- YARV VM Generator helps us
  - Ruby to “Pre-compiled”
  - Ruby to “C”

- Purpose
  - Eliminate Loading-Time
  - More aggressive optimization
  - Obfuscation (?)
[PLAN]

Multi-VM Creation

- Purpose
  - Embed Ruby into Application
    - mod_ruby, ...
  - Sand-box
Multi-VM Overview

Ruby (YARV)

VM1

RT

RT

... RT

VM2

RT

Native Thread
System S/W

NT

NT

... NT

Thread Scheduler

S/W

Processor(s)

PE

PE

... PE

H/W

PE: Processor Element, UL: User Level, KL: Kernel Level
Multi-VM Points

- How to control VMs?
  - C Level? → Designed with Nobuyoshi Nakada
    - Making new VM is need only 3 lines
  - Ruby Level?

- How to share environments Inter VM
  - Trade off between Isolation and Util.
[PLAN]
Customizable VM Core

- Ruby is toooooo FAT to use XXX purpose
  - Many Many Convenience Methods/Feature
- Need Re-design Ruby Core
[PLAN]

Debug/Profile Support Feature

- Only cheap Debugger/Profiler API
  - `set_trace_func`, `Thread#set_trace_func`
- Introduce “break” instruction?
Future Work

Benchmark

- Current Benchmark suits is for checking YARV Performance
  - Focus to YARV optimization
  - Toy benchmarks
- We need more pragmatic benchmarks
Summary

- YARV Merged into Ruby 1.9
- I’m working at Advanced VM Topics
  - Performance
    - Parallel Thread Execution
    - Embedding Float Value
    - JIT Compiler
    - Pre-Compiler
      - “Ruby to Compiled file” Compiler
      - “Ruby to C” Compiler
  - New Feature
    - Multi-VM Creation
    - Customizable Ruby Core
    - Debug/Profile support feature
Merging YARV is not a goal, but a start

VM is a very flexible infrastructure to hack
One more thing…
I’ll make a laboratory from 2009, Apr
- Department of Creative Informatics, Graduate School of Information Science and Technology, The University of Tokyo
- Graduate School
- Lab is at Akihabara, Tokyo, Japan

Unfortunately I can’t employ you as Research Assistant
- There are not enough grants in Japan…
Research Topics

- Ruby, Ruby, Ruby, PHP, Ruby, Ruby
- Ruby, Ruby, Ruby, Ruby, Python, Ruby
- Ruby, Perl, Ruby, Ruby, Ruby, Ruby
- Ruby, Ruby, Ruby, Ruby, Lua, Ruby
- Ruby, Ruby, Java, Ruby, Ruby, Ruby
- Implementation of Programming Language
- Operating System / Processor Architecture
- Software development
if you.have_interest(
  :Japan, Tokyo, :Akihabara,
  :Japanese,
  :Ruby, :Research, :Development
)
  you.send_mail_to "ko1 at atdot dot dot net"
end
Thank you for your attention!
Any Questions?

ささだ こういち
Koichi Sasada
Ko1 at atdot dot dot net

Ruby Meets VM, Koichi Sasada, EURUKO 2008