Can you see me?
Future of Ruby VM
Talk about Ruby VM Performance.

Ruby VMの未来，とかなんとか

SASADA Koichi <ko1@rvm.jp>
Department of Creative Informatics,
Graduate School of Science and Technology,
The University of Tokyo
Summary of My Talk

“Scaling Ruby (without the Rails)” Seems Interesting!

“Monkeybars: easy cross platform GUIs” Also Does!

On My Performance Interesting, Former is Preferred 😊 Anyone make a Log?
Summary

CRuby/YARV is **NOT a “BEST” Solution** for Ruby VM Performance.

However, CRuby/YARV is **“GOOD” Enough Solution** for Us, the Pragmatic Ruby Programmers, at least **Several Years**.
Self Introduction

Recent Report about Me

- **ko1 - Koichi (Given Name) Sasada (Family Name)**
  - From Japan, 5\(^{th}\) RubyConf since 2004, 4\(^{th}\) Speak
  - YARV Developer

- **Lecturer**
  - Department of Creative Informatics, Graduate School of Science and Technology, The University of Tokyo.
  - Lecture: Programming System, but only 3 students attend

- **SASADA-lab**
  - If you want to research about Ruby or Virtual Machine, Systems Software in Japan, please contact me.
  - 2 students are there, but no one want to hack YARV.
Caution! (re-re-re-review)

- I can’t speak English well
  - If I say strange English, you can see the slide page
    - Or ask another Japanese. They can speak English well.
    - My Slides uses Small Characters (against Takahashi-san’s Presentation Method)
- If you have any question, ask me with:
  - Japanese (recommended)
  - Ruby, C, Scheme, Java, …, Python, Haskell, …
  - Or Easy English
Agenda

• Perspective of Ruby VM Performance
  • VM Performance Discussion
  • Our Performance Policy

• Introduction of Our Research
  • Hidden Optimization Techs.
  • Ricsin Project
  • Ruby to C AOT Compiler Project
  • atomic-Ruby Project
  • MVM Project

• Summary
Remember The Evan’s Classification

JRuby is for Java Programmers
IronRuby is for .Net Programmers
Rubinius is for Ruby Programmers

CRuby is for C Programmers
OK.
Let’s Talk about the “C”,
The Benefits and Limitation
Evolution of VM Performance
My Prediction

Good at First
Now, CRuby is Good one
Finally, Rubinius is Best for Ruby’s Performance
CRuby has Limitation
Question: When get here?

Time / Effort / Money
Techniques for VM Performance

- Simple Optimization Techniques
  - C-level VM Techniques

- Advanced Optimization Techniques
  - Dynamic Code Generation
    - Speed-up using Native Machine code Compiler
    - Just in Time Compilation
    - Polymorphic Inline Cache
    - Selective Inlining
  - Online Feedback Optimization
    - HotSpot JIT Compiler
    - Tracing JIT
Pros and Cons of JRuby/IronRuby

- Using Awesome VM
- Pros.
  - Many Clever People Working on each VM
  - No Code is Good Code.
    - No Bugs are Generated.
  - Many Libraries on Each Environments
  - Easy (?) to Use Parallelization
- Cons.
  - Not Only Focused on Ruby, Semantics Gap
  - Can’t Use C Extensions Directly
Pros and Cons of Rubinius

- Most of Code is Written in Ruby
  - Like Java
- Pros.
  - Ruby in Ruby
    - Meta-Circular Interpreter
  - **Best Way** to Improve Performance in the Long Run
    Because They Can Analyze Most of Programs.
  - Mainly Focus on Ruby
- Cons.
  - **Long Way** to Get High Performance VM
Pros of “C” Ruby

- **Portability**
  - Most of Environments have GCC Porting.
- **Maintainability**
  - Everyone Know C.
- **Extensibility**
  - Easy to Write Extension with C.
- **Performance Improvement**
  - Easy to Write Simple (Machine Independent) Optimization.
Cons (Limitation) of “C” Ruby

- C Extension Libraries or Methods written in C
  - GC Problem
    - Conservative Mark & Sweep Stop The World GC
  - Inlining Problem
    - Can’t Inline C code into Ruby Code
  - Limitation of Program Analysis
Our Performance Policy

- CRuby is Not “Best” Solution but “Good” One
- Continue to Improve CRuby’s Implementation
  - in C
  - in Machine Dependent Way
- Pragmatic, Practical Selection
  - at least several years
Keywords for Success

- “Embedding”
- Parallelization
Introduction of Our Research

- To Take Advantage of “C”, Some Projects are Running
  - Hidden Optimization Techs on YARV
  - Ricsin: Mix-in C to Ruby Project
  - Ruby to C AOT Compiler Project
  - atomic-Ruby Project
  - Multi-VM Project
Hidden/Left Optimization Techs

- Turned Off on 1.9.1 by Default
  - Tail call Optimization
  - Optimization using Unification
  - Stack Caching
- Left Easy Optimization
  - Efficient Method Caching
  - Efficient Fiber Implementation using Platform dependent way such as makecontext()
- These Optimizations will be Merged into 1.9.2
Ricsin: Mix-in C to Ruby

- Embed a part of C Program into Ruby
- Like an RubyInline, but Embed Directly
- Usage Example
  - Use C Libs Directly
  - Replace All Built-in Classes/Methods
  - Test Ruby C APIs
  - Performance Improvement Continuously
def open_fd(path) # Ruby
  fd = __C__(%q{
/* C */
  return INT2FIX(open(RSTRING_PTR(path), O_RDONLY));
})
raise 'open error' if fd == -1
yield fd
ensure
  raise 'close error' if -1 == __C__(%q{
/* C */
  return INT2FIX(close(FIX2INT(fd)));
})
end
Ricsin Total View

- **rcb File (Ruby+C)**
- **Ricsin Translator**
  - **Makefile**
  - **C File (C)**
  - **rb File (Ruby)**
    - **so File (C Extension)**
      - Needed for Execution
  - **C Build Env. (C compiler, etc)**

Load/Exec

C Build Env.

Ricsin-Ver CRuby

CRubyNeeded for Execution
/* A Part of Generated C Source */
#define v (cfp->lfp[3])
#define r (cfp->lfp[2])
VALUE ricsin_func_1(
    rb_control_frame_t *cfp)
{
    const VALUE self = cfp->self;
    {
        /* Embed C Body */
        rb_p(self); /* show "main" */
        return INT2FIX(FIX2INT(v) + 1);
    }
    return Qnil;
}
#undef v
#undef r

/* Embed C Body */
rb_p(self); /* show "main" */
return INT2FIX(FIX2INT(v) + 1);

const VALUE self = cfp->self;
{
    /* Embed C Body */
    rb_p(self);
    return INT2FIX(FIX2INT(v) + 1);
}
return Qnil;

#undef v
#undef r
Ricsin: Evaluation

- Performance Evaluation (Not a Usability)
- Evaluation Environment
  - Env.1: Intel Xeon E5335, Linux
  - Env.2: SPARC T2, SunOS 5.10
- Evaluation Items
  1. Calling C Function (null call)
  2. Example on Iterator
  3. Matrix Multiprior
### Ricsin Evaluation of Calling Null Function

- Calling Null C Function
  - Null C Method
  - Null __C__ Embed

<table>
<thead>
<tr>
<th>Environment</th>
<th>C (sec)</th>
<th>Ricsin (sec)</th>
<th>C/Ricsin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Env.1 (Intel)</td>
<td>0.44</td>
<td>0.05</td>
<td>8.8</td>
</tr>
<tr>
<td>Env.2 (SPARC)</td>
<td>4.56</td>
<td>0.44</td>
<td>10.4</td>
</tr>
</tbody>
</table>
Ricsin
Evaluation: Iterator Optimization

- Rewrite Iterators with Ricsin
  - C: Current Iterator
  - Ricsin: Rewriting with __Ccont__
  - Ruby: Rewriting with Pure Ruby

Env.1 (Intel)  Env.2 (SPARC)
Ricsin Evaluation: Matrix Multiplier

- Matrix Multiplier with Fixnum Elements
- Replace 12 Lines Ruby Code to 36 Lines C Code Directly

<table>
<thead>
<tr>
<th>Environment</th>
<th>Ruby (sec)</th>
<th>Ricsin (sec)</th>
<th>Ruby/Ricsin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Env.1 (Intel)</td>
<td>10.57</td>
<td>0.57</td>
<td>20.33</td>
</tr>
<tr>
<td>Env.2 (SPARC)</td>
<td>85.31</td>
<td>6.73</td>
<td>12.68</td>
</tr>
</tbody>
</table>
svn co
http://svn.ruby-lang.org/repos/ruby/branches/ricsin
Ruby to C AOT Compiler

- Translate Ruby Script to C Source Code at Ahead of Time
  - Compile Ruby to Bytecode
  - Translate Bytecode to C Source Code
- Performance Improvement by
  - Eliminate VM Instruction Dispatch
  - Optimization by C Compiler
  - Eliminate Parse/Compile Time
Ruby to C AOT Compiler

• Ahead of Time Compilation
  1. Compile Ruby Script to VM Bytecode
  2. VM Bytecode to C

Ruby script → VM Bytecode → AOT compiler

→ Native code

C compiler → C source code
Ruby to C AOT Compiler

- Execution with Ruby VM

Native code (AOTed) → VM Insns → Ruby script

evaled String → VM Insns → Extension written in C
### Evaluation Environment

<table>
<thead>
<tr>
<th>Env</th>
<th>CPU</th>
<th>Memory</th>
<th>OS</th>
<th>C Compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td>32bit Linux</td>
<td>Intel PentiumD 2.80GHz</td>
<td>2 GB</td>
<td>Linux 2.6.24</td>
<td>gcc 4.2.3</td>
</tr>
<tr>
<td>64bit Linux</td>
<td>Intel Xeon 3060 2.40GHz</td>
<td>1 GB</td>
<td>Linux 2.6.18</td>
<td>gcc 4.1.2</td>
</tr>
<tr>
<td>cygwin</td>
<td>Intel Core Duo U2400 1.06GHz</td>
<td>1.5 GB</td>
<td>Windows Vista SP1</td>
<td>gcc 3.4.4</td>
</tr>
<tr>
<td>PS3</td>
<td>Cell Broadband Engine 3.2GHz</td>
<td>256 MB</td>
<td>Linux 2.6.16</td>
<td>gcc 4.1.1</td>
</tr>
</tbody>
</table>
Ruby to C AOT Compiler Evaluation Results

<table>
<thead>
<tr>
<th>Scenario</th>
<th>32bit Linux</th>
<th>64bit Linux</th>
<th>cygwin</th>
<th>PS3</th>
</tr>
</thead>
<tbody>
<tr>
<td>while loop</td>
<td>5.26</td>
<td>5.17</td>
<td>3.40</td>
<td>2.85</td>
</tr>
<tr>
<td>fibonacci number</td>
<td>1.66</td>
<td>1.38</td>
<td>1.46</td>
<td>1.34</td>
</tr>
<tr>
<td>pentomino</td>
<td>1.44</td>
<td>1.26</td>
<td>1.02</td>
<td>1.36</td>
</tr>
</tbody>
</table>

Speedup Ratio

2008/12/11
Future of Ruby VM - RubyConf2008
Related Work

- ruby2c by Eric, Ryan
  - Subset Ruby to C
- yajit by Shinh
  - JIT (yarv bytecode to IA-32 with Xbyak)
- yarv2llvm by Miura-san
  - JIT (yarv bytecode to LLVM asm)
atomic-Ruby Project

• Issue: Ruby is too Fat
  • Involves Convenient Functions.
  • Complex and Rational will be Built-in at Ruby 1.9
  → Difficult to Use “Embedded” Environment

• “Embedded”
  • Embedded System such as Resource Limitation Devs.
    • In Many Case, Numeric Tower or m17n are not needed.
  • Application Embedded Ruby
    • Application needs “DSL Engine”, doesn’t Full-set Ruby
atomic-Ruby Project (cont.)

- We Need Slim Ruby Interpreter
- atomic-Ruby makes “Suitable Ruby Interpreter”
  - Ruby Interpreter for Application
  - Ruby Interpreter for Environment (such as Embedded Systems)
  - Ruby Interpreter for Driver Application
- Utilize CRuby’s Portability
- 3 Sub-Project with 3 Students
  - Plug-in/out Built-in Classes/Methods
  - Pre-Compilation and Remove Parser/Compiler
  - Switch Core-Feature such as GC, Regex, Thread, etc
atomic-Ruby
Incremental GC

- Switch GC Algorithm
- Mark Partially
  - Execute App and Mark partially
  - Reduce Application Stop Time
Auto Write Barrier Detection

- Write Barrier is Needed for Several GC Algorithms.
  - Need Interpreter and Extensions.
  - Need Special Knowledge of VM and GC.
  - Cause Critical Bugs if WB Insertion Miss.

- Automatically WB Detection System
• Stop Time of Application (Mark Phase)
  • Insert Many WBs.
By The Way, Other CRuby GC Related Projects

- Generational GC (Kiyama)
- 1 bit Reference Count GC (Matz)
- Floating as Special Constant (ko1)
- Lazy Sweep (autherNari)
- Bitmap GC (Enterprise Ruby, autherNari)
- Mostly Copying GC (Ugawa)
Multi-VM (MVM) Project

- Multi Virtual Machine in One Process
- Each VMs are able to run in Parallel
  - Each VMs have Giant VM Lock.
- High Speed Inter-VM Communication
  - Inner Process Communication
Multi-VM Overview

Ruby (YARV) VM1 VM2

Native Thread System S/W

Thread Scheduler

Processor(s)

PE: Processor Element, UL: User Level, KL: Kernel Level
Multi-VM (MVM) Project

Sponsored by
Sun Microsystems, Inc.

Nobu (a.k.a Patch Monster) is Working for This Project
MVM

svn co
http://svn.ruby-lang.org/repos/ruby/branches/mvm
Summary

CRuby/YARV is
NOT “BEST” Solution
for Performance.

However, CRuby/YARV is
“GOOD” Solution for Us,
the Pragmatic Ruby Programmers,
 at least Several Years.
Summary (cont.)

- CRuby is Enable to Evolve Moreover
- Some Projects to Take advantage of CRuby
  - Ricsin: mix-in C to Ruby Project
  - Ruby to C AOT Compiler Project
  - atomic-Ruby Project
  - Multi-VM Project
Thank You for Your Attention.
Any Questions?

SASADA Koichi
<k01@rvm.jp>
Department of Creative Informatics,
Graduate School of Science and Technology,
The University of Tokyo
Accepted Method:
Ruby Thread and Native Thread (1:1) ← Ruby 1.9/YARV

PE: Processor Element, UL: User Level, KL: Kernel Level
Evaluation Result (Micro-benchmark)

x 7.4 w/8 Cores
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
Evaluation
Toy-Program

- Reduce Mem Time
- Encode/Decode don’t affect to Performance
Evaluation
Compared with other Ruby Impl.

From Comp. Lang. Shootout [4]

Mandel: 1.46
N-body: 1.39
Partial-sums: 1.54

Speedup Ratio (compared with 1.9.0)