YARV: Yet Another RubyVM

Innovating the Ruby Interpreter

Koichi Sasada
Graduate School of Technology,
Tokyo University of Agriculture and Technology
2-24-16 Nakacho, Koganei-shi, Tokyo, Japan.
sasada@namikilab.tuat.ac.jp

ABSTRACT

Ruby - an Object-Oriented scripting language - is used worldwide because of its ease of use. However, the current interpreter is slow. To solve this problem, some virtual machines were developed, but none with adequate performance or functionality. To fill this gap, I have developed a Ruby interpreter called *YARV (Yet Another Ruby VM)*. YARV is based on a stack machine architecture and features optimizations for high speed execution of Ruby programs. In this poster, I introduce the Ruby programming language, discuss certain characteristics of Ruby from the aspect of a Ruby interpreter implementer, and explain methods of implementation and optimization. Benchmark results are given at the end.

Keywords

Interpreter Implementation, Scripting Language, Ruby

1. INTRODUCTION

Ruby is the interpreted scripting language developed by Yukihiro Matsumoto for quick and easy object-oriented programming[2, 7]. It is simple, straight-forward, extensible, and portable. It has many features to process text files and to do system management tasks (as in Perl[1]) and many more.

Ruby has following characteristics.

- Simple syntax
- Normal OO features (class, method call, etc.)
- Advanced OO features (all values are objects, Min-in, Singleton method, etc.)
- Dynamic-typing, re-definable behavior, dynamic evaluation
- Operator overloading
- Exception handling
- Closure and method invocation with a block
- Garbage collection support
- Dynamic module loading
- Many useful libraries
- Highly portable

However, current Ruby intepreter is slow. This is because current interpreter (old-ruby) works by traversing abstract syntax tree and evaluates each node. To solve this problem, I have developed new Ruby interpreter called YARV (Yet Another RubyVM), which is a stack machine and runs Ruby programs in compiled intemediate representation of sequential instructions. I'm working to replacing old-ruby with YARV.

This poster is dedicated to discussing the Ruby programming language and the advantages and challenges that Ruby presents as an interpreter target, with speed-up being the principal goal. YARV's implementation and optimization features are then presented and the results are evaluated.

2. YARV IMPLEMENTATION

2.1 Overview

YARV is a simple stack machine written in C. The VM has a stack, a program counter (PC), a stack pointer (SP), some frame pointers (FP). YARV compiles a Ruby script into YARV instruction (intermediate) code sequences. The instruction set is designed for Ruby specifically.

YARV reuses many parts of old-ruby, namely the Ruby script parser, the object management mechanism, the garbage collector and more. In fact, YARV is implemented as an extension module for old-ruby.

YARV currently works on Linux with GCC and Windows2000/XP with Visual C++ or cygwin.

2.2 Intepreter Auto Generation

To create the virtual machine, I generate the code for the VM from a VM description written in a VM description language (VMDL) like vmgen[5]. Figure 1 shows the definition of an VM instruction (named `instruction1`).

```
DEFINE_INSTRUCTION
instruction1 // instruction name
(VALUE op1)  // operand values
(VALUE sp1)  // popped values from stack
(VALUE r1)   // values will be pushed to stack
{
  // instruction logic of instruction1
  // using op1, sp1
  // and at last assign value to r1
}
```

Figure 1: VM description language

In the VMDL, one declares operands, stack operands, and return values for each instruction. The programmer doesn't need to write the code to control the PC, SP, stack or fetching operands.

Furthermore, VMDL can generate optimized code automatically. The topic will be treated in the next subsection.

2.3 Optimization

YARV was implemented with many optimization techniques in order to create a high-speed interpreter.

Instruction dispatch makes use of dynamic threaded code[3] with GCC's extended feature (label as value) instead of `switch/case` statements in C.

Since all values in Ruby are objects, Ruby has no primitive types. For example the Ruby program `1+2` actually means `1.+(2)` (a method `+` is sent to reciever `1` with an argument `2`). To maximize efficiency, some methods are compiled to specialized instructions. In this case, method `+` is compiled to the spezialized specialised instruction `opt_plus`. `opt_plus` checks the receiver (self) and the argument. If they are both Fixnums, it checks whether method `+` of class Fixnum has been redefined or not. If it has not been redefined, it adds these values and pushes the result onto the stack. Otherwise, the normal method dispatch sequence is performed.

Operands unification and instructions unification (also known as super instruction) is used to optimize. If the programmer specifies that an instruction with specific operands or instruction sequence should be unified, the VM generation system generates unified instructions and compiler logic for this instruction.

YARV supports 2-level (2 registers, 5 states) static stack caching[4]. The VM generation system generates stack caching instructions and translater for compilation.

Ahead-of-Time (AOT) compilation of Ruby programs is also supported. The AOT compiler translates a Ruby program to a C program which runs on YARV. The C compiler then generates native machine code that is more efficient than YARV instruction code.

3. EVALUATION

Table 1 shows running time of benchmarks on old-ruby and YARV. This results were evaluated on Pentium-M 1.2Ghz, 1024MB memory, Windows XP and cygwin, gcc 3.4.4.

Table 1: Benchmark results

Benchmark	Ruby (sec)	YARV (sec)	Ruby/YARV
ackermann	29.86	2.61	11.4
Fibonacci	12.72	1.83	7.0
tak	17.36	2.41	7.2
matrix	3.92	1.40	2.8
sieve	10.45	1.81	5.7
count_words	0.69	0.63	1.1
whileloop	16.22	0.99	16.4

4. CONCLUSION

In this extended abstract, I described characteristics of the Ruby language and a new implementation of the Ruby interpreter called YARV. Many interpreter optimization schemes have been applied in the creation of YARV that have caused speed-up compared to old-ruby.

In the current Ruby interpreter, Ruby's multi-thread system is supported in user-level. This means that we can not write true parallel application in Ruby. As a solution to this problem, YARV will support native threads (Operating System managed it). This will enable Ruby programs to be more scalable.

Also I am planning to desgin Multi-VM instances mechanism like Java Multi-Talking VM[6]. This will improve performance and assist application programs that embed a Ruby interpreter.

Ultimately, I will replace current Ruby interpreter with YARV and YARV becomes The Ruby Virtual Machine.

Acknowledgement

This project is assisted by Exploratory Software Project 2004 (youth) and 2005 from IPA (Information-technology Promotion Agency, Japan) .

I want to address of thanks to Michael Neumann, Daniel Amelang, yarv-devel mailing list members and all Rubyists.

5. REFERENCES

[1] Perl.com: The source for perl – perl development, perl conferences. http://www.perl.com/.

[2] Ruby home page. http://www.ruby-lang.org/en/.

[3] A. Ertl. Threaded code. http://www.complang.tuwien.ac.at /forth/threaded-code.html.

[4] M. A. Ertl. Stack caching for interpreters. In *Proceedings of the ACM SIGPLAN 1995 conference on Programming language design and implementation*, pages 315–327. ACM Press, 1995.

[5] M. A. Ertl, D. Gregg, A. Krall, and B. Paysan. vmgen: a generator of efficient virtual machine interpreters. *Softw. Pract. Exper.*, 32(3):265–294, 2002.

[6] J. J. Heiss. The multi-tasking virtual machine: Building a highly scalable jvm. http://java.sun.com/developer/technicalArticles/Programming/mvm/, 2005.

[7] D. Thomas, C. Fowler, and A. Hunt. *Programming Ruby.* The Pragmatic Programmers, 2004.