2015 15th International Conference on Computational Science and Its Applications

mruby – Rapid Software Development for Embedded Systems

Kazuaki Tanaka
Faculty of Computer Science and Systems Engineering
Kyushu Institute of Technology
Fukuoka, JAPAN
Email: kazuaki@mse.kyutech.ac.jp

Avinash Dev Nagumanthri
Graduate School of Computer Science and Systems Engineering
Kyushu Institute of Technology
Fukuoka, JAPAN
Email: avinash@sein.mse.kyutech.ac.jp

Yukihiro Matsumoto
Fellow
Network Applied Communication Laboratory Ltd.
Shimane, JAPAN
Email: matz@ruby-lang.org

***Abstract*—In order to improve the development efficiency of embedded software, we have developed a programming language called mruby. We apply the object-oriented programming language Ruby to embedded system development. As compared to the interpreter of Ruby, mruby programs are executed by the compiler and VM. The memory footprint of mruby VM is sufficiently small, Ruby program is able to be executed on a limited resource device.**

In this paper, we show the benefits of using mruby in embedded software development. We also showcase the features and the development environment of mruby, and implementation results of simple application.

***Keywords*-Ruby; embedded software; virtual machine;**

I. Introduction

An embedded system is a computer system with a larger mechanical or electrical system built on hardware and software. The feature of electrical products is complicated and the importance of the software in modern embedded systems is increasing. On the other hand, the development cost of the software is also on a rise.

We propose an environment to reduce the cost of software development in this paper. By reducing the process time in software development by virture it automatically increases the productivity of software. We mainly focus on the programming aspect, by applying object-oriented programming language Ruby on embedded systems development.

mruby is a lightweight programming language of Ruby. The key point of this environment is execution of a program. Ruby is an interpreter, and mruby is compiler on virtual machine(VM). Since the VM virtualizes the device architecture of embedded systems, we can develop embedded software under device independent environment.

The name ‘mruby’ stands for embedded Ruby. In natural sense, the name should be eruby but eruby is used in HTML embedded Ruby, so we have choosen ‘mruby’ which emphasize eMbedded Ruby.

II. Embedded Systems

In the field of modern industry, product cost reduction is a necessity. Industrial equipments and electric home appliances with a dedicated function within a larger mechanical or electrical system, often with real-time computing constraints are called embedded systems. Embedded system consists of implementation of software on hardware. By complicating the function of the product, development cost of software increases. The increase of software development cost is a problem, as the product price reflects on the development expenditure. It is said that approximately 50% of the production cost contribute to software development [1].

A purpose of embedded system is to control hardware by program code. Real-time control is indispensable for hardware control. The real-time control is a mechanism to guarantee the processing time when a program controls hardware. Because software controls hardware, the access of the hardware layer is necessary.

In order to reduce the cost of the product, resources of the hardware device is limited to minimum required. Processing power and memory size of the device is smaller than a general-purpose PC. Limiting memory and processor performance, not only reduces the cost, also contributes to power saving. Therefore, this limit is appropriate and natural in embedded system development.

In recent years, the implementation of complex software is required under the constraints of hardware resources. Recent embedded systems provide complex functionality. In the implementation of complex software, it is natural to use a higher-level programming language. To efficiently develop complex software, language with a higher level of abstraction is necessary.

III. Ruby for Embedded Software Development

A. Ruby

Ruby is a pure object-oriented language that was released in 1995 by Yukihiro Matsumoto as an open source software,

978-1-4673-7367-8/15 $31.00 © 2015 IEEE
DOI 10.1109/ICCSA.2015.22

it is widely known by its development efficiency[2]. Ruby code is simple and flexible description with high readability. This advantage causes software maintenability to be increased drastically. In particular, Ruby on Rails, which is a web application framework published in 2004, Ruby has often come to be used in web application development. Ruby on Rails is taking the advantage of Ruby's flexibility and provides efficiency for application development.

Ruby has become an international standard in 2012 as ISO 30170. In addition, it has been standardized as JIS X 3017 (Japanese Industrial Standards).

B. mruby

The programming language mruby is a lightweight Ruby. mruby has the features of Ruby, and is a development tool that has been optimized for embedded systems. mruby is also compliant with ISO 30170. Because the syntax and grammar of the programming language are exactly same as Ruby, mruby has high efficiency and flexibility in software development. On the other hand, Ruby is an interpreter, which requires more resources for executing a program, however the resources available on the device are limited in the case of embedded systems.

Ruby program is executed by the interpreter. Execution of the program by the interpreter contains the following five processes; lexical analysis, syntax analysis, semantic analysis, optimization and execution. The first four of the five processes, does not depend on the execution environment. Execution process depends on the CPU architecture and memory architecture.

The process in Ruby interpreter is separated into a compiler part and a VM part in mruby. mruby compiler is a process that does not depend on the architecture of the device, with the support of lexical analysis, syntax analysis, semantic analysis, and optimization. The VM is a process of execution that depend on the device architecture. This is similar to Java compiler and JVM in Java language.

By mruby compiler and mruby VM is separated, the memory required during program execution is reduced. First, the mruby compiler converts Ruby program to mruby " byte code". The byte code is an intermediate code which is independent from the device architecture. Even if the byte code is output as a file format, it is not affected by the byte alignment. This means that developers can freely choose the environment by which the byte code is generated.

mruby VM executes sequentially mruby byte code. The mruby byte code is a kind of register transfer language. Therefore, the byte code is a sequence of instruction set for the operation of registers. For mruby VM details are shown in the later section.

C. mruby application development

mruby provides an efficient development environment for embedded systems. Developing traditional embedded

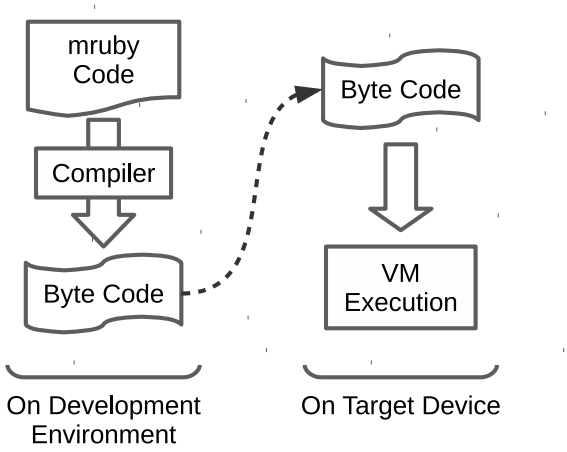

Figure 1. Application development by mruby

system, we must prepare a specific development environment corresponding to the target device. If the target device is subject to change or the software needs to be deployed to other devices with change in their development environment, it is necessary to prepare another development environment to support the new target device.

Development on different development environments is costly. Since the program implementation is distinct with the new development environment, the cost is proportional with reference to the new development environment instances. Software developed on different development environments, must be individual validated on each environment.

Procedure of application development in mruby is shown in Figure 1. Environment for application development, it's execution environment is different. This is the same with reference to cross-development environment of embedded software development. However, while the cross development environment is an environment that is dependent on each device architecture, development environment of mruby is independent from individual architecture.

IV. Virtual Machine Implementation

A. VM Architecture

The mruby VM sequentially executes the byte code. Byte code is a sequence of 32-bit fixed instruction set. Each word has a 7-bit opcode and 25-bit operands (Figure 2). VM is implemented as a register transfer language (RTL). In the current implementation of mruby (ver 1.1.0), about 90 of the instruction sets are defined[3]. Each instruction set is to transfer the registers, which is up to 128 registers, or to control the VM status.

Table I shows a part of the opcode.

The mruby VM has been specializing in the execution of object-oriented language. All registers are managing objects. The type of opcode of Table I is similar to the general CPU architecture of the machine language. However, subroutine

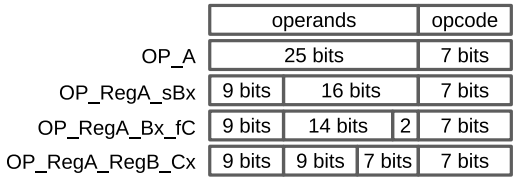

Figure 2. mruby instruction word

Table I
A PART OF THE OPCODE

opcode	description
OP_NOP	(wait for a cycle)
OP_STOP	(suspend running VM)
OP_MOVE	Reg[A] := Reg[B]
OP_LOADI	Reg[A] := sBx
OP_JMP	PC := PC + sBx
OP_ADD	Reg[A] := Reg[A] + Reg[A+1]
OP_SEND	Reg[A] := Reg[A].Sym[B]

call in machine language is implemented as a method call in an object-oriented language. For instance, OP_SEND, which appears in Table I, hasn't been implemented as a subroutine call, but a message passing in an object.

mruby is also a dynamic typing language, this is to evaluate the objects and methods at runtime. Assume method in an object stored in a variable is called method table, which manages an associative memory of method name and its entry point. Methods in the class to which the variable is referenced, and is selected by the method name as a key.

B. Garbage Collection (GC)

Many scripting languages such as Ruby usually perform memory management with garbage collection, mruby also has the same mechanism. Garbage collector collects the used memory block, and allocates a new object on the collected memory block, which is reusable memory. 'Mark&Sweep' garbage collection method, whose throughput is relatively high, has been used in many languages with Ruby included.

Since the memory block is to determine whether in use, program execution is paused while the garbage collection proceeds to collect memory block. If the system is fast enough in processing and memory access by the CPU in the manner that of a general purpose PC, a normal garbage collection is accepted. However, in real-time embedded software, pausing of program execution causes a problem(Figure 3).

mruby Garbage collection without long pausing is called incremental garbage collection. Incremental garbage collection is to limit the size of memory block that needs to be collected at a time. mruby VM proceed with the execution of byte code and garbage collection in concurrent.

The amount of memory for the execution of mruby program when making use of incremental garbage collection is shown in Figure 4. Amount of memory Mark&Sweep garbage collection and incremental garbage collection is the

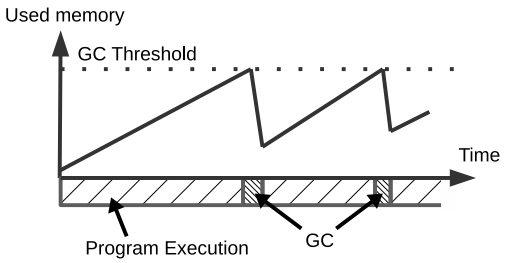

Figure 3. Mark&Sweep Garbage Collection

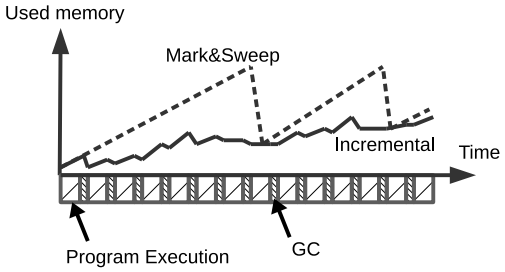

Figure 4. Incremental Garbage Collection

same. Using incremental method, since the execution and garbage collection are interleaved, there is no long pause.

Incremental garbage collector adds a color to the memory object. One of three colors is assigned to each object which are as follows:

White The object is not marked.
Gray The object is marked, but the linked objects are not marked.
Black The object is marked, and all the linked objects are also marked.

'White' color is assigned to all objects immediately to those which are allocated. Process of incremental garbage collection is similar to the Mark&Sweep method. First, garbage collector marks 'gray' to the root object, and then, recursively marks 'gray' to the objects that are linked to the root object.

In recursive marking process, if the object which is only links to marked are then marked as 'Black' object. 'Black' objects need not to be spread for recursive marking process. After all in recursively marking process, garbage collector finds that 'white' objects are not in use and the garbage collector collects them.

C. C Function Call

C language or assembler is used in coding of the lower layer to operate the device directly. Since mruby byte code is executed by the VM, it must call in the driver written in C through the VM from mruby. However the memory space of the VM is managed by the garbage collector, driver written in C is using the memory management mechanism

Table II
MRUBY RELATED PROGRAMS

program name	description
mrbc	mruby compiler
mruby	mruby VM
mrdb	command line debugger
mirb	mruby interactive command interface

```
def fib(n)
  if n<2 then
    1
  else
    fib(n-1)+fib(n-2)
  end
end
```

mruby source code

```
1 000 OP_ENTER      1:0:0:0:0:0:0
2 001 OP_MOVE       R3      R1
2 002 OP_LOADI      R4      2
2 003 OP_LT         R3      :<      1
2 004 OP_JMPNOT     R3      007
3 005 OP_LOADI      R3      1
3 006 OP_JMP                016
5 007 OP_LOADSELF   R3
5 008 OP_MOVE       R4      R1
5 009 OP_SUBI       R4      :-      1
5 010 OP_SEND       R3      :fib    1
5 011 OP_LOADSELF   R4
5 012 OP_MOVE       R5      R1
5 013 OP_SUBI       R5      :-      2
5 014 OP_SEND       R4      :fib    1
5 015 OP_ADD        R3      :+      1
5 016 OP_RETURN     R3      return
```

mruby bytecode

Figure 5. mruby code and compiled bytecode

Table III
TEST ENVIRONMENTS FOR MRUBY VM

environment name	CPU/RAM size
Raspberry Pi	ARM Cortex-A7 / 512MB
Beagle Bone Black	ARM Cortex-A8 / 512MB
GR-SAKURA	Renesas RX63N / 128KB
General Purpose PC	Intel Core 2 Duo / 4GB

```
1 def fib(n)
2   if n<2 then
3     1
4   else
5     fib(n-1)+fib(n-2)
6   end
7 end
8
9 puts fib(35)
```

Figure 6. fibonacci sequence

of the C library. The memory blocks allocated by the driver, because it is outside the scope of garbage collection, they may sometimes cause memory leaks.

The mruby VM have APIs for allocating the memory block for garbage collection. When allocating the memory blocks are referenced inside mruby on to the driver. These APIs are very useful. In addition, mruby also has APIs to control the behavior of garbage collection. The APIs provides the feature of enable/disable garbage collection, execution, and configure the pausing time in incremental garbage collection.

V. EXPERIMENTS AND RESULTS

A. mruby compiler and VM

We developed mruby related programs in Table II. All programs are released as open source softwares[4] and mruby and its documents are released under MIT license.

A bytecode is generated by compilation of a ruby program shown in figure 5. Each line of mruby code(Figure 5 left) is properly compiled into mruby bytecode(Figure 5 right). For readability, bytecodes are shown in mnemonic representation, its original generated bytecode length is 221 bytes.

The each column of bytecode shows 'line number of mruby source code', 'line number of mruby bytecode', 'op code' and 'operands'. mruby VM is specialized for ruby program execution. Since Ruby is a pure object oriented language, all operations are method calling. For instance, the line 003 of bytecode in Figure 5 is a comparison between two integers. The '<' method is called with an operand instead of a comparison operator.

This bytecode should be executed on mruby VM. mruby VM is device dependent program, which is cross-compiled for each target device. The cross-compilation of mruby VM is necessary but is easy because the mruby is implemented based on C99 standards. We tested on Raspberry Pi, Beagle Bone Black, GR-SAKURA and General Purpose PC(Table III). In GR-SAKURA test environment, where in the memory size is 1 MB of Flash ROM. The mruby VM program is stored in Flash ROM instead of RAM.

We got same results on all individual mruby VM when the mruby bytecode was executed.

B. Application 1: Fibonacci Sequence

We measured the consumption of memory size and execution time of mruby executi on operation. To compare with Ruby program, we executed them on general purpose PC because Ruby interpreter can only be executed on PCs. The PC has intel Core 2 Duo CPU@1.8GHz with 4GB memory.

Some fibonacci number was calculated by Ruby/mruby program(Figure 6). The execution time and consuming memory size is shown in Table V. The algorithm of fibonacci sequence calculation is recursive function call. By this experiment we can evaluate deep function call and optimization.

About the execution time, we can see the startup time and the speed of execution. Because Ruby program has

Table IV
EXECUTION TIME AND CONSUMING MEMORY OF n-TH FIBONACCI SEQUENCE

	Ruby		muby	
n	time[s]	memory[KB]	time[s]	memory[KB]
15	0.11	2,984	0.03	1,480
20	0.12	6,864	0.05	1,480
25	0.15	6,864	0.31	1,480
30	0.74	6,864	3.03	1,480
35	6.97	6,892	34.11	1,480

```
1  n = 1000000
2  while n>0 do
3    a = Array.new
4    a << n
5    n -= 1
6  end
```

Figure 7. GC test

Table V
EXECUTION TIME AND CONSUMING MEMORY OF GC TEST

	Ruby		muby	
iterations	time[s]	memory[KB]	time[s]	memory[KB]
100	0.11	6,992	0.03	1,480
1,000	0.11	7,056	0.03	1,480
10,000	0.11	7,152	0.04	1,480
100,000	0.14	7,872	0.13	1,480
1,000,000	0.26	6,992	0.96	1,480

optimization and execution cache technology during execution operation , the execution performance of Ruby is higher than mruby. mruby bytecode is pre-compiled and the size of bytecode is small. The mruby VM just loads the bytecode and execute the opcodes sequentially. As the result the startup time of mruby execution operation is shorter than Ruby.

About consumption of memory, we can clearly watch the memory footprint. In Table V, the consuming memory contains both mruby VM program code and application program code. The mruby VM is small and its footprint is also small. The consuming memory size is constant even though n is different. This result shows that the consuming call stack for deep function call is comparatively small.

C. Application 2: GC

We tested the GC in mruby VM by executing a mruby program shown in Figure 7. The execution result of Figure 7 is meaningless but at execution time many array objects are generated and wasted.

The GC collects the waste array object and reuse the released memory block.

At the 3rd line of Figure 7, an array object is created, and inserts an integer object into this array object by the 4th line. Because the created array object is overridden by next iteration, the previous array objects are wasted. Since GC properly collects the garbage objects, we can see the consumption of memory size almost constant at execution time.

The startup time of mruby VM is faster than that of Ruby, which is same as previous result.

D. Application 3: Hardware Access

Figure 8 shows a mruby program which controls hardware. The GPIO access methods are similar to the libraries

```
1  Gpio.pinMode 0, Gpio::OUTPUT
2  loop do
3    sleep 0.5
4    Gpio.digitalWrite 0, Gpio::HIGH
5    sleep 0.5
6    Gpio.digitalWrite 0, Gpio::LOW
7  end
```

Figure 8. Hardware access

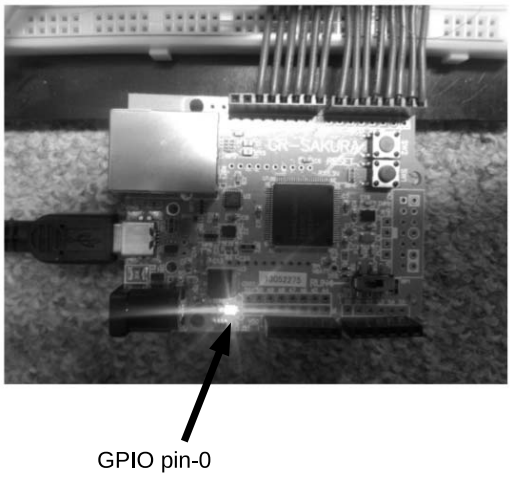

Figure 9. mruby on GR-SAKURA

that of Arduino. This program controls the output to that of pin-0 of GPIO (General Purpose I/O). The digitalWrite method in GPIO class is used in 4th and 6th line of Figure 8. This methods requires two parameters, which is the pin number to control and the output value of the pin. Since a LED is connected to pin-0, we can check mruby's behaviour by viewing the blinks of the connected LED.

This program was executed on target devices shown in Table III. An environment on GR-SAKURA is shown in Figure 9.

Of course the GPIO access is different between each device and general purpose PC does not have such GPIO pins to be controlled. mruby VM hides the hardware dependencies. We only implement a code using GPIO methods, which is implemented in mruby VM.

In mruby VM for PC, the digitalWrite methods is not to control the signal but output the current value of pins. This mechanism works as a debugging environment, by which we can analyse the behavior of mruby program corresponding to the developing environment(Figure 10).

VI. CONCLUSION

With the help of mruby, software development environment that is independent of the device architecture was realized. We can run one mruby program on various devices

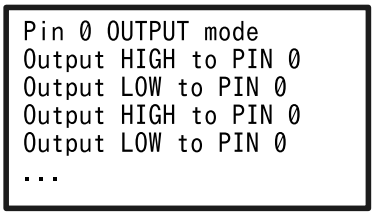

Figure 10. mruby on PC

with varied architecture. We simply code a mruby program and the program can be executed on different devices, which have different specification I/Os without recompilation. The only device dependent point is the VM, because the interface to call the C driver functions is different between devices. But once we build a VM for one target device, we can use the same VM among different applications on that device.

The mruby has been released as open source software. We are hoping that mruby can contribute to the efficiency of embedded system development and bring many advantages for application developers.

ACKNOWLEDGMENT

This research is supported by JSPS KAKENHI Grant Number 25330065.

REFERENCES

[1] Hiroshi MONDEN, Kiichiro TAMARU, “Embedded System development Process Reference guide”, IPA 2012

[2] The Ruby Community, Ruby Programming Language, http://www.ruby-lang.org/en

[3] NPO mruby forum, mruby documentations–mruby ver.1.1.0, http://forum.mruby.org/, 2014

[4] mruby GitHub repository, https://github.com/mruby/mruby