

Hardware Acceleration of Language Processing in Scripting Programming Languages

Hiroyuki Maeda(✉) and Kazuaki Tanaka

Kyushu Institute of Technology, Kitakyushu, Japan
maeda@sein.mse.kyutech.ac.jp, kazuaki@mse.kyutech.ac.jp

Abstract. This paper describes a hardware acceleration of language processing system in scripting programming languages. Scripting programming languages are high development efficiency since which is easy to write and high source code readability. On the other hand, embedded software developments take a lot of time and the difficulty level is high. We think that it is good to use scripting programming languages in embedded software developments for the improvement of development efficiency.

However, scripting programming languages require many hardware resource such memory and their execution speeds are slower than procedural languages such as C language. Therefore, we propose a hardware acceleration of language processing in scripting programming languages. Hardware acceleration technology can improve the processing performance of scripting programming languages by a pipeline processing.

We focus on a scripting programming language 'mruby/c' for embedded software developments. We implemented the hardware which performs the memory management and the instruction decoder in mruby/c. In this paper describes results of implementation of the hardware acceleration and the performance evaluation.

Keywords: Scripting language · FPGA · Hardware acceleration

1 Introduction

Embedded systems are advancing high functionality and complication. Embedded system developments take a lot of time and the difficulty level of developments are high since embedded system developments require the both of hardware and software development. Generally, the cost of software developments take rather than the hardware. Embedded software developments require to control the hardware such as some sensors and output devices.

In addition, IoT (Internet of Things) which various devices are connected to the internet and are controlled and observed through the communication is

© Springer Nature Switzerland AG 2019
S. Misra et al. (Eds.): ICCSA 2019, LNCS 11621, pp. 407–416, 2019.
https://doi.org/10.1007/978-3-030-24302-9_29

used in many industrial areas. It is increasing the additional value of embedded system by IoT appearance. IoT developments require the developments of wireless communication techniques in addition to both of hardware and software developments. Embedded software developments including IoT take a long time and costs.

Therefore, we think that it is good to use a scripting programming language in embedded software developments to improve software developments efficiency. Many script programming languages enable easy programming. The number of codes of script programming languages are shorter than the procedural language such as C language and also superior in readability. It is possible to improve the productivity of software developments using script programming languages. It is also easy to learn scripting programming languages for programming beginners.

However, scripting programming languages require many memory for the execution but hardware resources in embedded system are limited. In addition, the execution speed of scripting programming languages are slower than the procedural language such as C language.

In this paper, we propose the hardware acceleration of language processing system in scripting programming languages. The execution speed of hardware processing by the pipeline processing is faster than software processing. The hardware acceleration is possible to reduce the memory footprint of scripting programming languages. We improved the processing performance of scripting programming languages by using hardware acceleration.

2 Scripting Programming Languages

Scripting programming languages are a programming language that makes it easy to write codes because it is a simple syntax and is superior in readability. Currently, many scripting programming languages have appeared and are used in some system developments. This section describes the features of scripting programming languages.

2.1 Scripting Programming Languages for Web Application Developments

Many scripting languages are mainly used in Web application developments. The programming language Ruby is one of scripting programming languages and used widely in Web application developments.

– **Ruby**

The scripting programming language ‘Ruby’ is popular in developing Web applications. Ruby is an object-oriented scripting programming language. Ruby is developed in JAPAN and released as an open source software. The Web application framework Ruby on Rails for Ruby is provided and is popular in many enterprises.

Ruby is a programming language with high development efficiency since it is easy to write programs and has high readability. Ruby is a simply syntax and the code of Ruby can be written shortly.

Ruby is an interpreter language and a dynamic programming language. The types of variable and functions are interpreted in execution. Thus, programmers need not to write types and Ruby can extend programs flexibly.

Ruby performs a dynamic memory allocation in the same as many scripting programming languages. Programmers are not required to consider the memory management since the memory allocation and free in execution are performed the garbage collection in Ruby's language processing system. Programmers are not required to consider the memory management since the memory management in execution are performed by the garbage collection in Ruby's language processing system.

Ruby is difficult to use in embedded software developments with a little hardware resources. Ruby assumes the execution environment which has many hardware resources such as a server machine. The Ruby interpreter needs a lot of hardware resources to execute syntax and semantic analysis process. In addition, execution speeds become slow by Ruby interpreter and dynamic memory allocation.

2.2 Scripting Programming Languages for Embedded Software Developments

Ruby has another implementation for embedded software developments.

– **mruby**

mruby is the lightweight implementation of Ruby for embedded software developments. It is published as an open source software [1]. mruby is enabled to execute on embedded devices while maintaining Ruby's high developments efficiency. Ruby is an interpreter language and is not suitable for embedded software developments. Therefore, mruby is a complier language and is executed by the mruby virtual machine (mruby VM). The construction of mruby program execution is shown in Fig. 1.

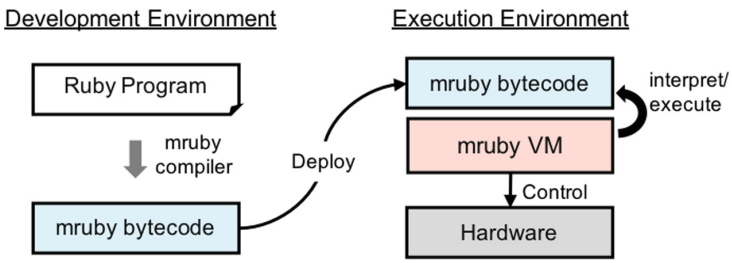

Fig. 1. mruby program execution

mruby is a compiler language and the dedicated mruby compiler generates from Ruby program to the mruby bytecode in the development environment. The mruby bytecode is a hardware independent format. The mruby VM interprets the mruby bytecode and executes in the execution environment. The mruby VM is consisted a part that executes mruby bytecode according to Ruby syntax and a part that controls hardware. Developer can rewrite a hardware dependent part in mruby VM and mruby can deploy in various embedded devices.

– **mruby/c**

mruby/c is the another implementation of mruby. mruby/c is published as an open source software [2]. mruby/c is the further lightweight implementation than mruby.

In the execution mechanism of mruby/c, the mruby compiler and the mruby bytecode is used same as mruby, but mruby/c is different the virtual machine. The mruby/c VM is a small implementation than the mruby VM. The mruby/c VM restricts some functions and reduces the memory size than mruby VM.

mruby and mruby/c are different the required memory size and target devices are different each. mruby requires less 400 KBytes memory and mainly targets embedded devices with OS (Operation System). On the other hand, mruby/c requires only 50 KBytes memory and mainly targets small embedded devices such as one-chip microcomputer, especially small IoT devices.

mruby/c supports concurrent execution functions without OS [3]. In IoT developments, it requires to control many sensors and communication technologies. By using multiprogramming, an algorithm in main and a program to control sensors can be developed independently and be improved the development efficiency.

In this study attempted to implement the hardware acceleration of mruby/c.

3 Hardware Acceleration

This section describes the difference between software and hardware processing.

– Software Processing
  The merit of software processing is to change a processing flexibly by rewriting the source code. However, the execution speed of software is slow because it interprets and executes single instruction one by one.
– Hardware Processing
  In the merit of hardware processing, the execution speed is fast because it can be pipeline processing by implementing the dedicated circuit. The hardware processing is good at simple arithmetic processing. The circuit scale becomes large when the complex algorithm including many conditional branches is implemented. The hardware acceleration should be implemented circuits which is a simple processing such as less conditional branches.

The hardware acceleration is expected to improve the performance by implementing the software processing which has become the overhead in systems [4]. In addition, Cooperating between hardware and software is can be maintained the flexibility of rewriting the processing [5].

We analyzed the mruby/c VM for efficiency implementation of hardware acceleration. The hardware implementation of all language processing in mruby/c VM takes a lot of costs. Therefore, we thought that it is good to implement hardware of only processing that is expected the improvement of processing by hardware acceleration.

mruby/c VM firstly loads the mruby bytecode and repeatedly performs instruction decode and instruction execution. In addition, mruby/c VM performs the dynamic memory allocation and memory free in execution. The following processing is performed in each phase.

1. Loading the mruby bytecode and Instruction Decode
   mruby bytecode is consisted of header information and several machine language instructions. mruby/c VM fetches and decodes one machine language instructions from the mruby bytecode.
2. Instruction execution
   mruby/c VM executes the instruction according to the instruction types.
3. Memory Management
   mruby/c VM performs the dynamic memory allocation in execution. mruby/c VM dynamically allocates the memory when the instruction execution requires the memory. mruby/c frequently occurs the dynamic memory allocation in execution.

We focus on the instruction decoder of the mruby bytecode. The instruction decoder repeatedly performs simple bit operations. We implemented the hardware acceleration of instruction decoder of mruby/c.

We also focus on the dynamic memory allocation. We think that it becomes the bottleneck in processing since mruby/c frequently occurs the dynamic memory allocation in execution. In the dynamic memory allocation, it determines memory to allocate by calculating from the requested size and state of free memory. We implemented the hardware acceleration of calculation the free memory area to allocate.

4 Hardware Acceleration Implementation

In this study used a FPGA (Field Programmable Gate Array) for the implementation of hardware acceleration. The FPGA is a rewritable integrated circuit and can flexibly configure system by rewriting its internal logic circuit. We used Terasic's DE0-Nano board with Intel's Cyclone IV (EP4CE22) FPGA chip. This FPGA chip has 22,320 logic elements and 594 Kbit SRAM.

For the implementation of hardware acceleration, we used Nios II system that is a soft-core processor for an Intel FPGA. The Nios II system can be constructed embedded processors in FPGA. The Nios II system is also provided

many peripheral circuits for configuration of embedded systems. It is possible to develop original embedded systems in FPGA by using Nios II embedded processor and peripheral circuits. Since the Nios II can be reconfigured the system flexibility, it is suitable for research and development of embedded systems [6].

The circuit configuration of the implemented Nios II system is shown in Fig. 2.

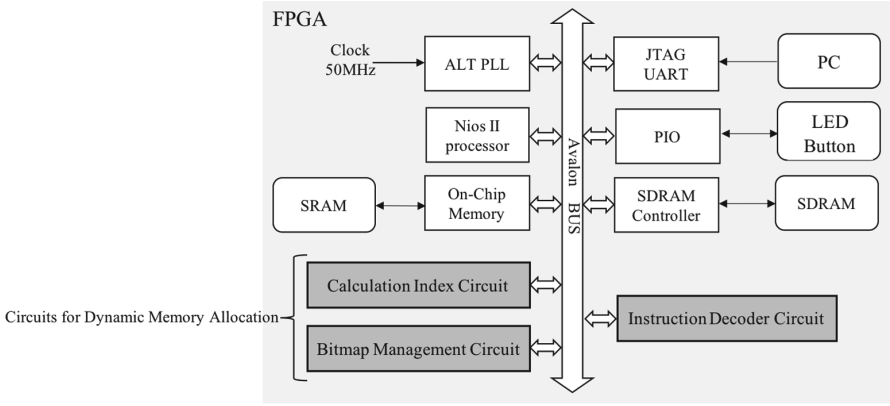

Fig. 2. Configuration of Nios II system

This system is constructed by Nios II processor, some peripheral circuits and custom component circuits. Their circuits are connected by the Avalon-Bus that is dedicated bus line. The details of each circuits are shown next.

- Nios II Processor : Nios II/f
  Nios II/f is a "fast" version designed for superior performance, which is the highest performance in the Nios II processor family. The wordlengh of Nios II processor is 32 bits. Nios II processor provides the software development environment based on GNU C/C++ tool chain.
- On Chip Memory (SRAM)
  The On Chip Memory is SRAM and an internal memory of the FPGA. The SRAM achieves a high speed access, but memory size is small capacity. The SRAM is stored the configuration information of circuit implemented in FPGA.
- SDRAM/SDRAM Controller
  The SDRAM is an external memory on FPGA board. The SDRAM is a large capacity but, a low access speed. The SDRAM is used as main memory and the mruby/c is written to it.
- ALT PLL
  The ALT PLL (Altera Phase-Locked Loop) multiples the input 50 MHz clock which is supplied from board to 100 MHz.

- JTAG UART
  The JTAG UART is the interface core that provides a serial communication between FPGA board and Nios II development environment on host PC.
- PIO
  The PIO (Parallel I/O) is the interface core that provides input/output access to user logic or external devices such as LEDs and switches.
- Instruction Decoder Circuit
  The Instruction Decoder Circuit is the custom component circuit which we implemented for the instruction decoder. The Instruction Decoder Circuit decodes instructions required mruby/c execution. These details describe in next section.
- Circuits for Dynamic Memory Allocation
  Circuits for Dynamic Memory Allocation are custom component circuits for dynamic memory allocation. Circuits for Dynamic Memory Allocation are configured the Index Calculation Circuit and the Bitmap Management Circuit in Fig. 2. These details describe in next section.

4.1 Custom Component Circuit of Instruction Decoder

We implemented the instruction decoder circuit that is a custom component circuit. The instruction decoder circuit can decode the mruby bytecode that mruby/c VM performs conventionally. The instruction decoder circuit is used the bit operations such as bit shift.

The wordlength of one instruction is 32 bits, which is consisted the operation code part and data part. Conventional, mruby/c VM decodes one instruction to the operation code and some data part one by one. The instruction decoder circuit decodes operation code part and some data part by pipeline processing.

4.2 Custom Component Circuits for Dynamic Memory Allocation

Custom component circuits for dynamic memory allocation are implemented the memory allocation function of mruby/c. mruby/c is implemented the TLSF (Two-Level Segregated Fit) memory allocator algorithm for dynamic memory allocation.

- TLSF Allocator
  TLSF allocator can be dynamic memory allocation that is fast that does not depend on data size and inhibits the fragmentation of memory [7]. TLSF is especially suitable to embedded systems that is time restriction such as real-time requirements. TLSF finely segregates and manages free memory blocks for each size by two-level. When TLSF allocates a memory block, it uses the index to each free blocks. TLSF manages free memory blocks of first and second level segregations by two bitmaps. TLSF calculates the index from required size and first and second level bitmaps. TLSF determines the allocation memory block from the index.

We implemented custom component circuits for the dynamic memory allocation. Custom component circuits for the dynamic memory allocation is consisted the Index Calculation Circuit and the Bitmap Management Circuit. The feature of custom component circuits for the dynamic memory allocation is shown in Fig. 3.

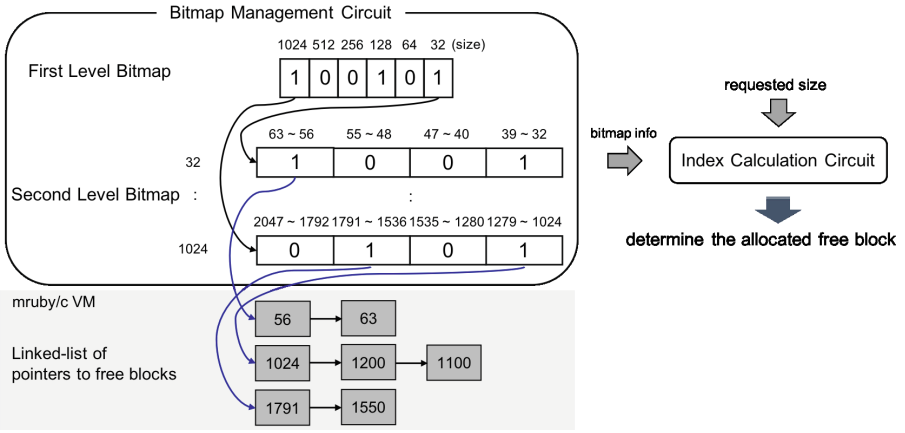

Fig. 3. Custom component circuits for dynamic memory allocation

The Bitmap Management Circuit manages bitmaps information of each size free memory blocks by two level. In first-level bitmap is segregated free blocks by a power of 2 size and set the corresponding bitmap to "1". In second-level is segregated the each size in first level more finely. When it determines the free memory block to allocate, it refers the bitmap information in Bitmap Management Circuit.

The Index Calculation Circuit is the circuit that calculates the index of free memory block to allocate. The Index Calculation Circuit calculates the index of free memory block to allocate from then requested size and the bitmap information.

mruby/c VM manages the pointer to the memory by the linked-list for each size. mruby/c VM cooperates the linked-list of free block and bitmap information in the Bitmap Management Circuit.

5 Evaluation

In this section describes implementation costs and the effect of the hardware acceleration of language processing in scripting programming language. The circuit scale of instruction decoder circuit and circuits for dynamic memory allocation are shown in Table 1.

The total circuit scale including NIos II system and custom component circuits is required 6,488 logic elements. Logic elements in DE0-Nano board is

Table 1. The number of logic elements

Total	Nios II system	Decoder circuit	Circuits for memory allocator
6,488	2,183	1,273	3,032

22,320, total scale of our configuration circuits is about 14%. The Nios II system without custom component circuits are required 2,183 logic elements. Instruction decoder circuits is required 1,273 logic elements. Custom component circuits for dynamic memory allocation is required 3,032 logic elements. Circuits for dynamic memory allocation are consisted index calculation circuit and the bitmap management circuit. The index calculation circuit is required 1,421 logic elements and the bitmap management circuit is required 1,611 logic elements. They are small implementation.

In their memory usage, mruby/c without custom component circuits is 151 KBytes. When we replaces memory allocation and instruction decoder with custom component circuits, its memory usage is 149 KBytes.

To confirm the effectiveness of hardware acceleration, we measured the execution speed of the only software processing and processing combining custom component circuits. In this study used program which combines calculating fibonacci numbers by recursion and calculating four arithmetic operations by while-loop as a benchmark program for measurements. Execution speeds are shown in Table 2.

Table 2. Execution speeds

Only software	With custom component circuits
1289 [ms]	1157 [ms]

In results of measurement, mruby/c that is only software processing takes time of 1289 ms. On the other hand, it takes time of 1175 ms using Custom Component Circuits for instruction decoder and memory allocation. Therefore, it was able to short 132 ms. It has been able to execute more fast by hardware acceleration.

6 Conclusion

The paper describes the hardware acceleration of a scripting programming language for embedded software developments. We analyzed the language processing of a scripting programming language and implemented hardware of language processing that it becomes the bottleneck in processing. In this study focused on a scripting language 'mruby/c' for small embedded system. For implementation of hardware acceleration, we used the Intel FPGA and the soft-core Nios II system. We implemented the instruction decoder circuit and circuits for dynamic

memory allocation. The circuit scale of the instruction decoder circuit and circuits for dynamic memory allocation are a small implementation. Results of investigation, we confirmed the effectiveness of hardware acceleration. mrubyh̨as been able to execute more fast by hardware acceleration.

Im future works, we will improve the execution speed of scripting programming languages by hardware acceleration. We think that the effect of hardware acceleration for scripting programming languages is low in current research results since execution speeds are slightly speedup by hardware acceleration. We will analyze the language system of scripting programming language and implement for more efficient hardware acceleration.

References

1. https://github.com/mruby/mruby
2. https://github.com/mrubyc/mrubyc
3. Tanaka, K., Maeda, H., Higashi, H.: Concurrent execution in scripting programming language 'mruby'. In: Gervasi, O., et al. (eds.) ICCSA 2018. LNCS, vol. 10962, pp. 136–146. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-95168-3_9
4. Jasrotia, K., Zhu, J.: Hardware implementation of a memory allocator. In: Proceedings Euromicro Symposium on Digital System Design. Architectures, Methods and Tools, pp. 355–358 (2002)
5. Joao, J.A., Mutlu, O., Patt, Y.N.: Flexible reference-counting-based hardware acceleration for garbage collection. In: ISCA 2009 Proceedings of the 36th Annual International Symposium on Computer Architecture, pp. 418–428 (2009)
6. Salunke, P.G., Sayyed, A.M.: Design of embedded web server based on NIOS-II soft core processor. In: International Conference on Electrical Electronics and Optimization Techniques (ICEEOT), pp. 488–492 (2016)
7. Masmano, M., Ripoll, I., Crespo, A., Real, J.: TLSF: a new dynamic memory allocator for real-time systems. In: Proceeding ECRTS 2004 Proceedings of the 16th Euromicro Conference on Real-Time Systems, pp. 79–86 (2004)