Concurrency in Go and Java: Performance Analysis

Naohiro Togashi
Software Engineering Lab
University of Aizu
Aizuwakamatsu, Fukushima, Japan
s1180107sh0stak0@gmail.com

Vitaly Klyuev
Senior Associate Professor
University of Aizu
Aizuwakamatsu, Fukushima, Japan
vkluev@u-aizu.ac.jp

***Abstract*— Go is a new programming language developed by Google. Although it is still young compared to other programming languages, it already has modern and powerful features inherited from existing programming languages, and some of these are similar to Java. Go is designed for quick time development. Concurrency is the one of the main its features. In this paper, we analyze the performance of Go, and compare it with Java from two aspects: compile time and concurrency. There are many studies about the performance analysis and comparison of programming languages, but only a few publications investigate Go. Some of Go performance evaluation are based on the experimental release of Go.**

To analyze concurrency features, we implement simple matrix multiplication programs in both Go and Java. Java implementation uses Java Thread, and Go implementation uses Goroutine and Channel. From the experiment, Go derived better performance than Java in both compile time and concurrency. Moreover, Go code shows the ease of concurrent programming. Go is still young, but we are convinced that Go will become the mainstream.

***Index Terms*— Go, Java, Concurrency, Performance, Evaluation**

I. Introduction

In 2009, Google introduced a new programming language called Go, which is a compiled, garbage-collected and concurrent programming language [1]. It is designed for fast compilation time, and ease of programming. Go is still not widely used compared to other programming languages, because only a few years have been passed after its first release. Go provides many modern and powerful features inherited from the existing programming languages, and some of these are similar to Java in its object-oriented programming, built-in concurrency mechanism, garbage-collection, and etc.

The Java Programming Language was released by Sun Microsystems in 1995 [2]. It is a concurrent, object-oriented, and garbage-collected programming language, which is widely used for several areas of application. Java has built-in support for concurrency: the `Thread` class, `Runnable` interface, and `java.util.concurrent` package. They provide powerful features for multi-thread programming.

In this paper, we focus on concurrency feature, which is the one of common features of Go and Java. The purpose of this paper is to analyze the performance of Go and compare it with Java on two aspects: compile time and the concurrency feature. To analyze the performance, we prepare simple matrix multiplication programs, implemented in Go and Java.

II. Related Work

There are many studies about the performance evaluation of programming languages. Some of the published papers are related to the Go language, but these results are based on the experimental release of Go [3] [4]. In 2012, Go version 1 was released. It contains several new language features and libraries. Additionally, a number of performance improvements are introduced. This is the first release of Go. So, it is important to analyze the latest Go, and compare it with other programming languages.

A. Loop Recognition in C++, Java, Go and Scala

Hundt reported the performance comparison between four languages - C++, Java, Go and Scala - [3]. Go derives better performance in terms of compile time. With respect to the run-time measurement, on the other hand, C++ derives the best performance. Hundt concludes that although Go provides interesting language features, the compilers for this language are still immature negatively influencing the performance.

B. Multi-Core Parallel Programming in Go

Tang presented two multi-core parallel programs in Go in order to show the ease of multi-core parallel programming using Go [4]. Implementations of benchmarks are parallel integration and parallel dynamic programming. He also measured performance of Go with shifting the number of cores used. He concludes that it is easy to write multi-core programs in Go, and the highest speed of benchmarks are derived on an octal-core AMD chip.

C. Ruby under Scanner: Comparison with Java

Das evaluated the performance of the Ruby language by comparing it with Java on the three aspects: language features, ease of programming, and performance in terms of time and memory [5]. In order to evaluate performance, he selected some sorting algorithms and matrix multiplication for evaluating threads. He concluded that Ruby is easier to

978-1-4799-4808-6 /14/$31.00 ©2014 IEEE

program and is much more compact, compared to Java, but Java is the preferred language for complex computation.

III. Method

This section explains the methods for performance comparison. Matrix multiplication is often used for performance evaluation of programming languages as shown in [5], [6], and [7]. In this experiment, we use simple matrix multiplication that is calculated by *C*=*AB*. To simplify the problem, we define two matrices that have same length of row and column. First, we divide matrix A into some processes, and calculate the product of a part of A and matrix B. After completion of all calculation, we combine them. Figure 1 illustrates the part of calculation of matrix C.

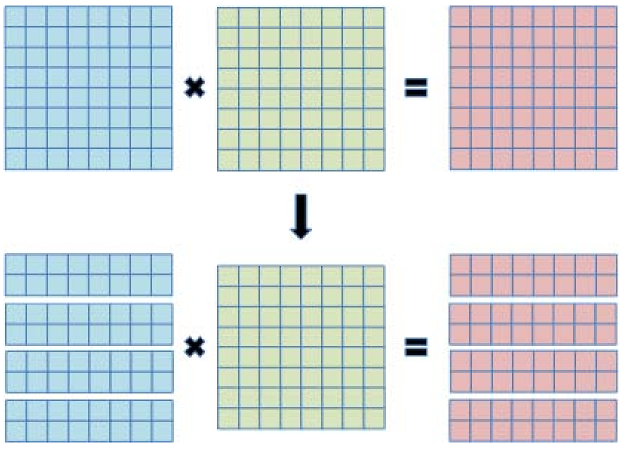

Fig. 1. Method of Matrix Multiplication (Example: 4 processes)

All the measurements were performed on the same machine. Details of the hardware and software are given below:

TABLE I
Hardware and Software Overview

Computer	MacBookPro8, 2
Processor	Intel Core i7, 2.2 GHz, Quad-Core (Hyper Threading: 8)
Memory	8GB
Operating System	Mac OS X 10.9
Go Compiler	go1.2
Java Compiler	java 1.7.0_45, Java SE Runtime Environment (build 1.7.0_45-b08, mixed mode)

The benchmarks are implementations of simple matrix multiplication in both Go and Java. In order to measure performance, we prepare four benchmark sources: `matrix.go`, `parallel_matrix.go`, `Matrix.java` and `ParallelMatrix.java`.

A. Compile Time

One of the purpose of the Go project is to build programs at high speed. Go provides a model that makes dependency analysis easy and avoids much of the overhead of C-style include files and libraries [8]. In the measurement, we use the Unix *time* command.

B. Concurrency

Java has built-in support for multi-thread programming. In Java, we can write concurrent programs by extending the `Thread` class or implementing the `Runnable` interface. On the other hand, Go provides a lightweight thread called Goroutine. Concurrent programming in Go is very simple. When we call a new goroutine, only we need to add the `go` keyword before the function call:

```
go function(arguments)
go func(parameters){body}(arguments)
```

Below we show the details of implementations to understand benchmarks clearly.

1) Java Implementation: In Java code of this experiment, we simply extend the `Thread` class, and override the run method:

```
class ParallelMatrix extends Thread{
   public void run(){
     /* implementation */
   }
}
```

Before the calculation, we get available CPUs on local machine by using `availableProcessors()` method to specify the number of processes.

```
NumOfThreads=Runtime.getRuntime()
      .availableProcessors();
```

After that, we use the `start()` method to start created threads, and we use the `join()` method to wait for finish of threads:

```
for(int i=0; i<NumOfThreads; i++){
   threads[i].start();
}

for(int i=0; i<NumOfThreads; i++){
   threads[i].join();
}
```

To measure elapsed time, we use the `nanoTime()` method managed by the `System` class. By using it, we can measure smaller time than using the `currentTimeMillis()` method.

```
long start=System.nanoTime();
    /* calculation */
long stop=System.nanoTime();
```

After the measurement, we calculate the difference between them.

2) Go Implementation: First, we need to specify the number of CPUs which is executed simultaneously, because Go uses only one thread by default. There are two ways to set the number of CPUs. The `GOMAXPROCS` environment variable limits the number of OS threads, and we can set `GOMAXPROCS` in the following way:

```
export GOMAXPROCS=8 #default is 1
```

We can also specify the number of CPUs by using the `GOMAXPROCS()` method managed by the `runtime` package in the Go code:

```
numCPUs:=runtime.NumCPU()
runtime.GOMAXPROCS(numCPUs)
```

`NumCPU()` is also managed by the `runtime` package, and returns the number of logical CPUs on the local machine, and `GOMAXPROCS()` sets the maximum number of CPUs. In the Go implementation, we define a `multiply()` function to calculate matrices, and the following is the part of implementation:

```
func multiply(arguments, ch chan)
   /* implementation */
   ch <- 0 // Send a value to notify
           // the end of goroutine.
```

The definition of the function above has a special value channel. `ch chan` is the declaration of a channel. It is a communication pipe which is based on the Communicating Sequential Processes (CSP) model. To wait for finishing goroutines, we use a channel. Creation of goroutines and waiting for finishing goroutines are following:

```
for i:=0; i<NumOfGoroutines; i++{
  go multiply( arguments )
}

for i:=0; i<NumOfGoroutines; i++{
  <- c // wait for
       // finishing goroutines
}
```

To measure elapsed time, we can use the way similar to Java. `Now()` is a method managed by the `time` package, and it returns current time:

```
start:=time.Now()
/* calculation */
stop:=time.Now()
```

After that, we use the `Sub()` method to get difference:

```
stop.Sub(start)
```

IV. Performance Analysis

All the benchmarks were run three times and we calculate the average value. Table 2 shows the number of lines and code size of benchmarks, and Table 3 shows the result of compile times. For both Go and Java, the latest compilers are evaluated. The *time* command outputs the *Real* time, *User* time and *System* time, and we take into consideration the *Real* time.

TABLE II
Number of Lines and Code Size

Benchmarks	Number of Lines	Code Size(byte)
matrix.go	40	743
Matrix.java	46	937
parallel_matrix.go	67	1283
ParallelMatrix.java	82	1979

TABLE III
Compile Time

Benchmarks	Compile Time (millisec)
matrix.go	0.214
Matrix.java	0.670
parallel_matrix.go	0.225
ParallelMatrix.java	0.694

Each benchmark contains same features. To measure code size, Unix *wc* command is used. As you see from Tables 2 and 3, code size of Go and Java have a little difference, but the compile time of Go is three times faster than that of Java.

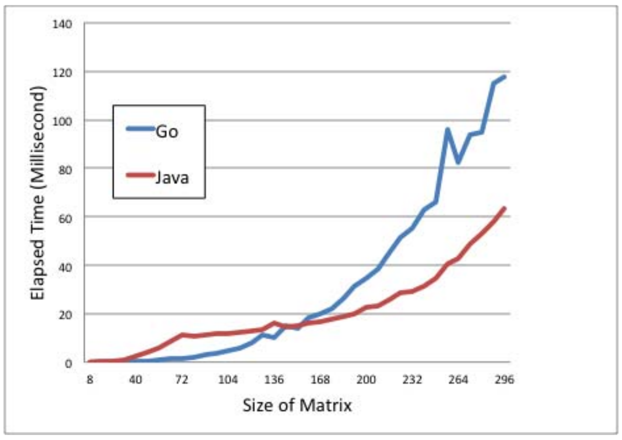

Fig. 2. Normal Matrix Multiplication

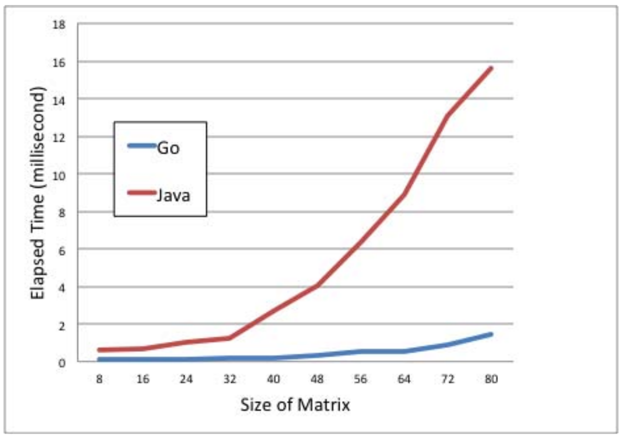

Fig. 3. Matrix Multiplication using Java Thread / Goroutine

The result of concurrency performance is shown in Figures 2 and 3. Comments to the rsults shown in Figure 2 are as follows, we measure and compare normal matrix multiplication of Go and Java, and we stop the measurement if the big difference occured in the result. You can see that the calculation time of Go rapidly increased compared to Java. Although it is said that Java is slower than other programming languages, Java performances were improved over many years. On the other hand, Go is also designed for fast execution speed, but Go is still young. Although Go version 1 was released, an immature part of Go influenced the result.

Figure 3 shows the result of matrix multiplication using Java Thread and Goroutine. We concentrate on evaluation of the overheads of adding Threads/Goroutines. In this experiment, Go derived better performance than Java. Java Thread provides powerful features for multi-thread programming, but their creation costs are very expensive.

V. Future Work

Go provides many packages related to networking. The `net` package provides an interface for network I/O, including TCP/IP and UDP, and it has some sub-packages. The `net/http` package provides basic HTTP client and server implementations, and provides features to handle HTTP requests. The `net/smtp` package provides features for parsing e-mail. Additionally, Go is supported by Google App Engine (GAE). GAE is a cloud platform for application development and hosting. It lets us develop, run, and deploy web applications on Google's infrastructure. So, Go is also suitable for building web applications on cloud environment. Cloud computing has been gaining popularity over the last few years. Developing web applications on cloud provides us many benefits: low costs, high-capacity storage, latest services, and more. As the future work, we are planning to develop a Schedule Management System using Go with GAE. Java is often used for building web applications, but simple syntax and fast build time of Go makes it more efficient to develop applications.

VI. Conclusion

In this paper, we implemented matrix multiplication programs in Go and Java to compare compile time and concurrency performance, and each benchmark is implemented in simple writing. Go derived better performance than Java in each experiment. Code sizes of two languages have a little defference, but compile time of Go is about three times faster than Java. It shows that Go makes dependency analysis easy, and we can develop applications efficiently by using Go. In the concurrency performance, Go derived quite better result than Java. We used Java Thread by extending the `Thread` class in the Java implementation. Java Thread is powerful, and provides useful features for multi-thread programming, but its cost of creation overhead is very high. Additionally, it is difficult to write correct multi-thread programs in Java, and its complicated design makes it hard to understand the concurrent mechanism. In the Go implementation, we used Goroutine and Channel. Unlike Java, Go takes a different approach to concurrency which is based on the CSP model. In Go, these mechanisms hide the complexity of concurrent processes. As you see from a part of Go code, it is very simple to wirte Go concurrent programs.

Only a few years passed after the release of Go, but it is updated frequently. It is still a young language, and contains some immatured parts, but Go has a potential to become one of the main programming languages.

Acknowlegement

This work was supported by JSPS KAKENHI Grant Number 25330410.

References

[1] The Go Programming Language Official Website, http://golang.org
[2] The History of Java Technology, www.oracle.com/technetwork/java/javase/overview/javahistory-index-198355.html
[3] R. Hundt, "Loop Recognition in C++/Java/Go/Scala", Google, Scala Days, 2011
[4] P. Tang, "Multi-Core Parallel Programming in Go", University of Arkansas at Little Rock, ACC'10, 2010
[5] S. Das, "Ruby under Scanner: Comparison with Java", University of California, technical report, 2010
[6] A. Pajjuri, H. Ahmed, "A Performance Analysis of Java and C", Columbia University, 2001
[7] J. L. Volakis, "A Comparison of Java, C/C++, and FORTRAN for Numerical Computing", University of Michigan, 1998
[8] The Go Programming Language - Frequently Asked Questions, http://golang.org/doc/faq [accessed on 25.12.2013]