POG0

David Thompson Rohit Saily Raymond Tu

April 2, 2020

Research Question

How does the compilation time of different programs using the P0 Compiler differ between a sequential implementation in Golang and a concurrent implementation in Golang that uses Goroutines to allow the Scanner to identify tokens while the Parser parses identified tokens?

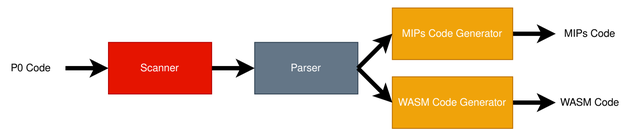
Resources

- For most of the implementation the IDE GoLand was used for the trans-compilation from the original P0 Compiler to Golang™
- GoLand was used to execute the parts of the compiler, conduct unit testing and exploratory testing
- For testing the research question, a Python script was ran to generate the test cases and run them automatically
- For documentation GoDoc was used to document all modules in the compiler

Obstacles Encountered

- Initially, Google™'s Grumpy transplier was going to be used to convert the original Python implementation to a Golang implementation. However, the transpiler outputted unreadable code organized in confusing modules and it seemed risky to move forward with that, so the original P0 implementation had to be manually transpiled. This took a lot of time and care since differences in Python and Golang had to be considered.
- Debugging runtime errors of the transpiled code was difficult due to the size of the P0 compiler and its use of recursion. To reduce the complexity of debugging, GoLand™'s built in debugger was used to set breakpoints and step through the compiler's execution.
- After completely transpiling the MIPS generator, it was found that the code could not finish generating. After debugging, it was determined that the correct code was being generated, as it appeared in memory, but generation failed towards the end for different test cases. This remains an unresolved mystery.

Architecture



Concurrency Implementation

```
reader := bufio.NewReader(f)
tokenChannel := make(chan SourceUnit, 500)
endChannel := make(chan int)
go ScannerInit(reader, tokenChannel)
go compileFile(tokenChannel, endChannel, destFilePath, "wat")
<-endChannel
```

Testing Plan

The general approach to testing was generating a large P0 program and seeing how long it took to compile the program with each of the three different implementations of the compiler. To generate a long program, a P0 program with two functions was written. In order to make the program longer, the function definitions were copied repeatedly, but with different names. Once a program of the desired length was created, it was compiled with each of the compilers 10 times. These compilations were timed. The average of these times was taken, then output to the console in a table format. This process was repeated for programs of different sizes. All of the above was automated using a Python script. This testing method measures the efficiency of the compilers while minimizing manual testing effort.

Development Statistics

File	Lines Of Code
p0-go-concurrent.go	61
CGmips.go	823
codegenerator.go	40
parser.go	803
scanner.go	252
symboltable.go	555
wasmgenerator.go	456

File	Unit Tests
scanner.go	44
symboltable.go	7

References

https://golang.org/doc/go1.9#parallel-compile

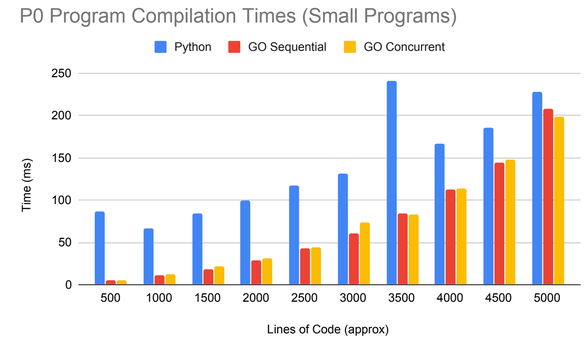
https://blog.golang.org/godoc-documenting-go-code

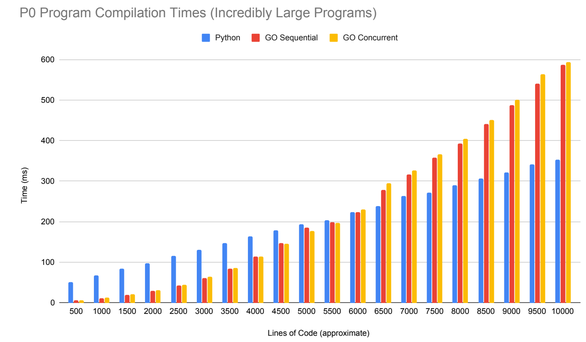
https://www.jetbrains.com/go/

https://www.cas.mcmaster.ca/~se3bb4/

Results

The following runtimes were generated on a Intel® Core™ i5-10210U CPU @ 1.60GHz 8 processor with 16 GB of RAM, running Fedora 31 (Linux).





Conclusions

In conclusion initially the GO compilers were faster than the Python, but with more lines of code the increase in speed decreased until the test were 6000 lines of code or higher, in which the Python compiler was faster than the GO compilers

The sequential GO compiler and concurrent GO compiler were roughly comparable in speed up to programs with 6000 lines of code, after which the sequential compiler is faster than the concurrent

It can be concluded that there is no benefit in speed with the implementation of the concurrent compiler, and with programs larger than 6000 lines of code it is detrimental. The overhead of the concurrent implementation is more expensive than any improvement in speed through multiple Goroutines. However, both implementations of the compiler in GO are faster than the Python with programs smaller than 6000 lines of code. Practically, most programs will not be larger than 6000 lines of code so there is potential application for the Go sequential compiler for some programs.