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?

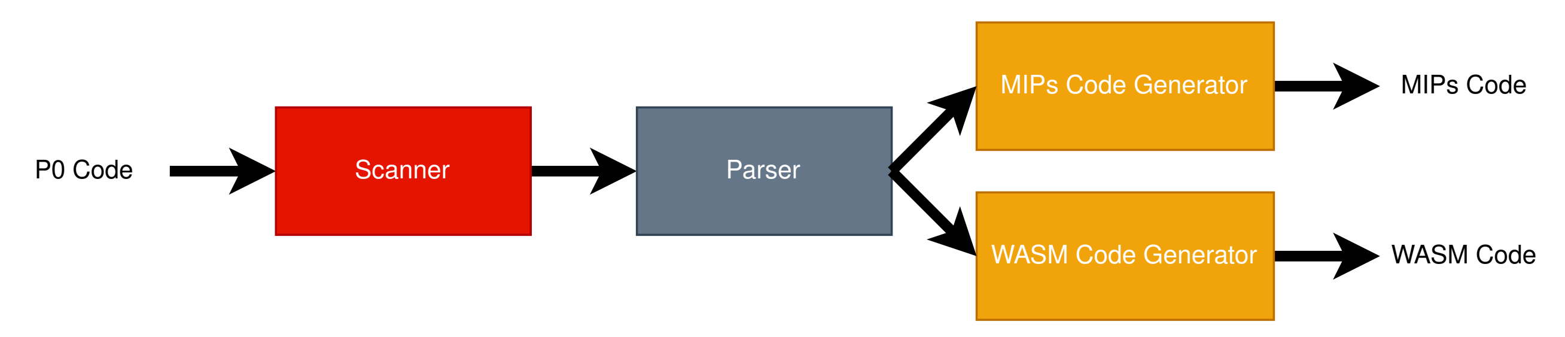
Resources

- The IDE GoLand was used to manually transpile P0 from Python to Golang and to develop the concurrent implementation
- GoLand was used to debug code, conduct unit tests, and carry out exploratory tests
- Python was used to create a script for generating test cases and running them all automatically
- GoDoc was used to document all modules of the compilers

Obstacles Encountered

- Google's Grumpy transplier was planned to be used to convert the original Python implementation to a Golang implementation. However, the Grumpy outputted unreadable code and presented its output in a cumbersome manner that would take a significant amount of time and resource to figure out. To get over this obstacle, the P0 implementation in Python had to be manually transpiled. This took a lot of time and care since significant differences in Python and Golang had to be considered throughout development.
- Debugging runtime errors of the manually transpiled code was difficult due to the size of the P0 compiler and its use of recursion. To reduce the complexity of resolving errors, 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 and it was being stored in memory; however, generation failed towards the end for all 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 goal of testing was to determine the efficiency of the compilers while minimizing manual testing effort.
- To make a set of test cases, a P0 program with a couple hundred lines of code was created as the first test case. Then, its function definitions were copied repeatedly and renamed to make valid programs that were also larger.
- Each test a total of 10 times and were automatically timed and averaged.

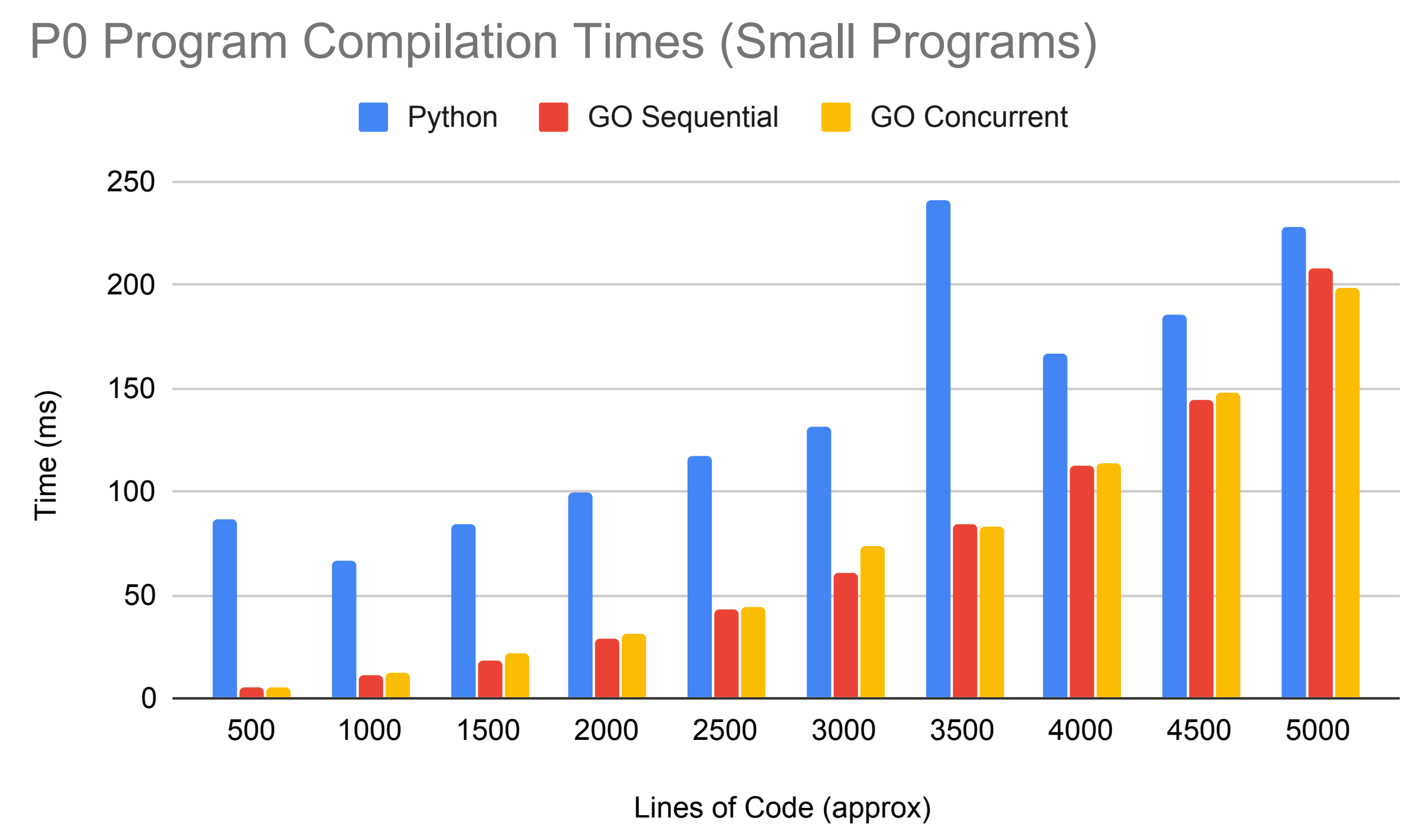
Development Statistics

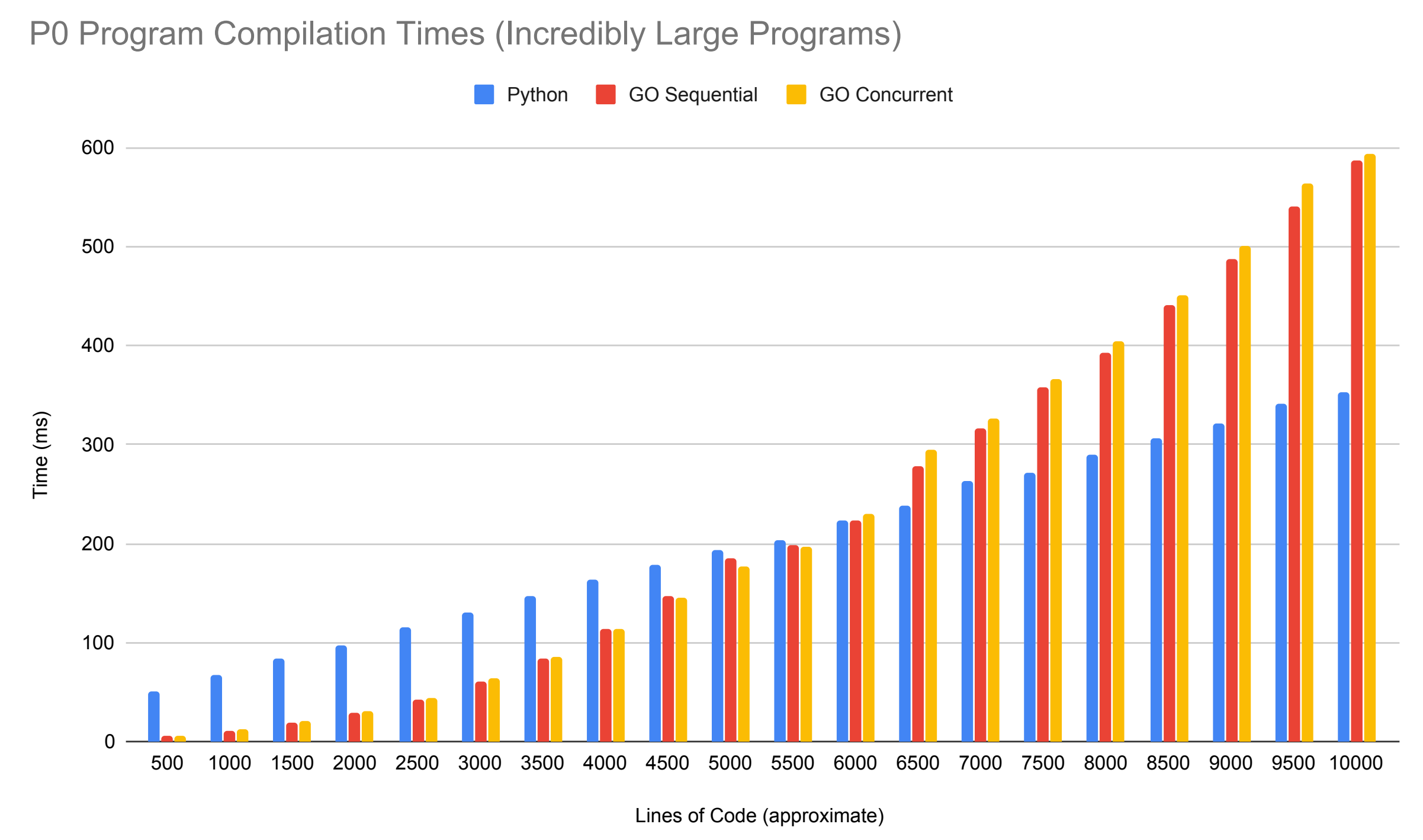
File	Lines Of Code
p0-go-concurrent.go	61
CGmips.go	823
codegenerator.go	40
parser.go	803
scanner.go	252
symboltable.go	555
wasmgenerator.go	456

File	Unit Tests
scanner.go	44
symboltable.go	7

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

- The results indicate that there is no benefit to implementing a concurrent compiler where the Scanner scans tokens while the Parser parses already scanned tokens.
- For programs that are larger than 6,000 lines of code, the difference in the measured time is relatively more noticeable.
- The difference is attributed to overhead of the concurrent implementation is more expensive than any improvement in speed through the Goroutines.

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/`