4TB3 Project Development Plan

David Thompson, 400117555   Rohit Saily, 400144124   Raymond Tu, 400137262   February 25, 2020

1   Project Description

1.1   Research Question

How does the compilation time of the P0 Compiler differ between a sequential implementation and a concurrent implementation using Goroutines in Golang?

1.2   Relevant Work

Many compilers forego concurrent compilation of parts of one file for two reasons. First, the concurrency mechanisms provided by the languages that other compilers are coded in have too much overhead. The time it takes to set up tasks to compile the code concurrently exceeds the time that could be saved by compiling concurrently. Next, it is much easier to compile a group of files concurrently, with one instance of the compiler for each file. This keeps each core of the processor occupied, so additional concurrency on top of this may not improve the compilation time much.

However, Go's compiler implements compiling different portions of one file concurrently. `https://golang.org/doc/go1.9`. It is a self-hosted compiler; it itself is written in Go. This means it can benefit from the cheap concurrency that the language provides in order to facilitate the concurrent compilation.

TODO: references

1.3   Developmental Challenges

When developing this project, a few challenges need to be overcome to ensure the project is completed correctly and on time.

One major challenge is making the only significant difference between the P0 compiler and its concurrent counterpart the fact that the first is a sequential program and the latter is a concurrent program. This is necessary to ensure that the only property being compared between them are the fact that the first is a sequential program and the latter is a concurrent program. This challenge will be overcome by:

1. Transcompiling the original P0 compiler from Python to Golang. This ensures that the compilers are compiled to machine code under the same principles.

2. Executing tests within the same runtime environment. This ensures that the compilers execute in the same way.

3. Testing the compilation time of both compilers for the same set of inputs. This allows the compilation times to be compared directly.

Another major challenge is designing the concurrent implementation of the P0 compiler.
First, designing a concurrent program is difficult in general since it requires consideration of possible interleavings of program instructions. This aspect of the challenge will be overcome by referencing material from McMaster University's SFWRENG 3BB4 course.
Second, making a *compiler* concurrent is challenging since various dependencies between statements need to be respected when separating the compilation into concurrent processes. To overcome this the program to be compiled will be decomposed into independent units which can be treated as their own programs and compiled concurrently.

1.4   Test Plan

We will develop a test suite of example programs which encompass the features of the P0 language. We will pull from the example programs in the notebooks, and write our own. We will also develop syntactically incorrect programs, or programs with type errors, in order to test our error detection and reporting.

Using these programs, we will separately test each of the following components:

- Scanner - Given the source code, is the expected sequence of tokens generated by the scanner?

- Parser - Given the tokens corresponding to the source code, does the generated AST correspond to the expected AST?

- Generators - **TODO:** I guess compare to the code generated by the original compiler for these languages?

We will also perform integration tests by running all the steps of the compiler at once, ensuring that code generated by our compiler is identical to that of the existing implementation.

In addition to the above, we will unit test the symbol table. This particular program is more suited to unit testing than the other components, because individual searches and additions can be made. For the other components, an entire valid program is expected as input.

1.5   Documentation

For the documentation of our project we will be using GoDoc. GoDoc is similar to documentation generators for other programming languages, for example JavaDoc for Java and Doxygen for Python. GoDoc generates documentation in PDF or HTML format from comments in the source code. Following the standard conventions for GoDoc will provide easily readable source code and good documentation for all methods implemented in the P0 Compiler. GoDoc is chosen because it is the standard documentation standard for Golang, which is the main programming language we will be using for this project. The conventions for a standard GoDoc comment to generate HTML is easy to read, and alone would already provide readable source code to any viewer. In conclusion, GoDoc is intuitive to use and similar to existing documentation standards we have experience with, which makes it the perfect choice for this project.

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

2   Resources

2.1   Transcompilation

Python is generally an interpreted language, but Go is compiled before execution. This means that the Go implementation of the P0 compiler would likely run faster, even if no concurrency is used. In order to get a fair comparison between the compilers, a few methods are available.

The existing P0 compiler could be manually rewritten to Go. This method will take a long time, and would leave less time for refining the concurrent Go implementation.

Our preferred route is to transpile the existing Python code to Go instead of manually translating the program. Google offers a Python to Go transpiler called Grumpy.

Another option would be to compile the Python P0 compiler and the concurrent Go P0 compiler to a common language, then compare their run times in that language. This is not ideal, because concurrency works differently between programming languages. Go is known for having very efficent concurrency compared to other languages. As a result, the efficiency of the concurrent Go implementation may be understated. It would be preferable to use one of the previous two alternatives.

2.2 Development

To meet the deadline of the project, an IDE will be used to make the development of the concurrent P0 compiler relatively easier. GoLand will be used as the IDE since it is designed for Golang and offers a wide variety of development features that automate many tedious programming tasks.

2.3 Testing

For testing for errors in the P0 Compiler in Go, unit testing will be conducted. This unit testing will be done in the standard testing package that is included in Go. This testing package also includes statement-based code coverage checking that will be useful as a metric for how well the code is tested. The difference between this package and other standard unit testing for other languages such as PyTest for Python is that it does not utilize assert statements. Instead an error is thrown after a boolean check coded by the programmer. This will allow the test package to detect multiple errors in the program, because while an assert statement would stop the program after the first error, the test package in Go will instead detect the error and continue running other test cases. In conclusion, using the Go test package we can eliminate errors in the sequential and concurrent implementation in Go before we test the differences in their runtimes and answer the defined research question.

For timing the execution time of the P0 Compiler, we can use a variety of different time libraries included in Python or Go. To be consistent, one timing library will be chosen and used for testing the execution time of both implementations of the P0 Compiler. This will ensure accurate results to draw a conclusion from to answer the research question. How these libraries will be used for testing will be elaborated on in the Test Plan section.

2.4 Poster

LaTeX

Some inspiration from Pinterest for the poster board: `https://www.pinterest.ca/m9914/latex-research-posters/`

Overleaf has templates for poster boards as well. `https://www.overleaf.com/latex/templates/tagged/poster`

Since golang implementes concurrency in its compiler, we can reference the compilers code and documentation for the poster board.

3 Division of Labour

4 Weekly Schedule

Week	Deliverables
Week 1	Development Plan and Proposal
	Setting Development Environment
Week 2	Transcompilation and Beginning of Development
Week 3	Working Prototype
Week 4	Testing and Refinement of Prototype and Documentation
Week 5	Poster Presentation Creation
Week 6	Poster Presentation Creation