Progress Report

Name:	**Dylan Moss**
Email:	**dm894@cam.ac.uk**
Project Title:	**Automatic Parallelisation using Effect Tracking and Cost Analysis**
Supervisor:	**Alan Mycroft**
DoS:	**Ramsey Faragher**
Overseers:	**Robert Mullins and Marcelo Fiore**

Summary

The aim of my project is to create an automatic parallelisation compiler from my custom sequential programming language to parallelised Go code. This is done through side-effect tracking to identify independent blocks of code, and by tracking the size of data types to approximate execution times. I have written 3500 lines of OCaml code and completed three of my five success criteria (designing and parsing my language, implementing side-effect tracking, and producing parallelised Go code), with one more close to completion (implementing cost analysis). I now realise that my final success criterion, a full evaluation of my compiler, seems inappropriate for a progress report (more details in paragraph 3); and so intend to complete a more streamlined evaluation instead. Overall, I am 2–3 weeks behind schedule and intend to complete my success criteria by 14th Feb, giving me enough time to work on my extensions and the evaluation.

Some changes have been made to the original timetable, I set aside some time at the beginning of project to develop a framework, which has aided the development of the rest of the project. This initially set me back three weeks, but the time saved using the frameworks has outweighed this delay. However, there has been one major unexpected difficulty in my project so far. The type inference algorithm I designed for tracking data type sizes required a large amount of manual effort to cover all edge cases. Fortunately, I noticed that incorporating a large part of one of my extensions into the core project early could help solve this problem and save me time overall. I have since successfully implemented this, and taking this into account I am only 1–2 weeks behind schedule.

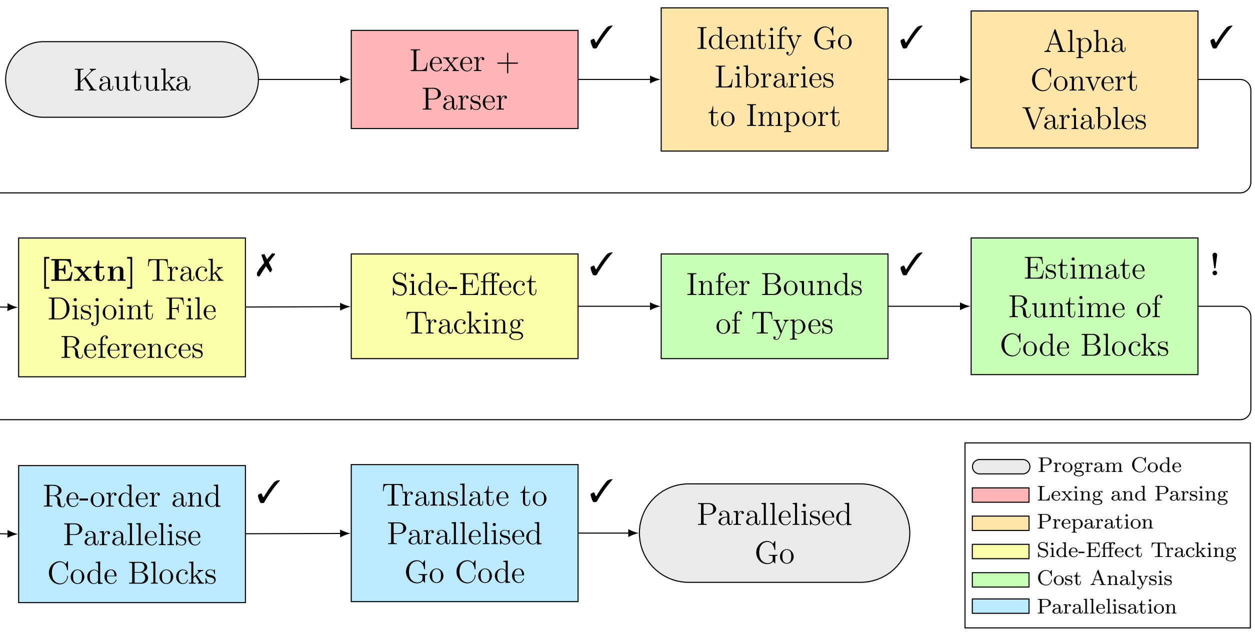
I have decided to change my final success criterion to be a more streamlined evaluation of the compiler, evaluating the quality of my generated code using synthetic benchmarks. A full evaluation determines whether I have met my success criteria, and so it would be inappropriate to include this in the success criteria list. I will carry out a full evaluation after I have completed some of my extensions, allowing for a richer evaluation as my language will become more expressive.

More details on my project's progress appear on the following page.

Details

The design of my compiler consists of multiple pipeline stages, described in the diagram below. I first designed the syntax for my Go-like custom language and built a lexer and parser for it. Next, I built the framework (described on the previous page), which allowed me to easily complete the *Preparation* stages of the pipeline. This involves identifying the Go standard libraries which need to be imported into the final Go code, and alpha converting all the program variables. I then carried out side-effect tracking for non-local variables — with added support for tracking console I/O and file system side effects to be included as part of my later extensions. I am currently working on cost analysis, for which I have completed inference for tracking the size of data types, but have yet to use this information to estimate the execution time of code blocks. The final stages of the pipeline, parallelising code blocks and translating the AST into parallelised Go, have already been completed.

My progress can be represented with the following diagram (✓ Completed, ✗ Not Started, **!** In Progress):



Kautuka
Lexer + Parser
✓
Identify Go Libraries to Import
✓
Alpha Convert Variables
✓
[Extn] Track Disjoint File References
✗
Side-Effect Tracking
✓
Infer Bounds of Types
✓
Estimate Runtime of Code Blocks
!
Re-order and Parallelise Code Blocks
✓
Translate to Parallelised Go Code
✓
Parallelised Go
Program Code
Lexing and Parsing
Preparation
Side-Effect Tracking
Cost Analysis
Parallelisation