Monash University

Electrical Engineering

Final Year Project (ECE4093)

Progress Report

Author:
James Anastasiou
ID: 23438940

Supervisor:
Dr. David Boland

May 24, 2016

Contents

1 Introduction

1.1 Purpose and Scope

This document aims to provide a summary of what has been achieved thus far in this final year project. This document seeks to be a supplement to the design document, which is a more in depth look at the current position and future predictions about the project. In order to best serve this purpose, this document includes:

- Project Description
- Project Status Overview
- Gant Chart

2 Project Description

My final year project aims to create a set of static code/compiler analytics tools to help determine which algorithms within a codebase may be easily parallelized. The specific intention it to provide users with a report describing which parts of their codebase may see a potential speed-up from redeveloping them as CUDA (GPU) kernels. In order to achieve this both benchmarking and theoretical analysis is required, as well as static analysis of the C description of the algorithm itself.

3 Major Accomplishments

The key accomplishments thus far in this project have been a combination of knowledge based as well as code output based. The beginning of this project was characterised by a steep learning curve, in order to write General Purpose GPU code to complete the project. Additionally the other component, that of the static code analyser also involved a steep learning curve, with further research and testing with the Clang Abstract Syntax Tree, through Clang LibTooling. Following these learning phases the major accomplishments of the project so far are:

4 Project Status

This section provides a summary of what has been completed so far within this FYP.

4.1 Completed Tasks

- Stable package and distribution setup
- Successful identification of simple parallel patterns
- Initial scaffolding of GPU benchmarking code
- Basic GPU peak memory performance benchmark
- Scaffolding of performance metric reporting
- Capable of analysing arbitrarily large C codebases

5 Current Position Review

This section aims to look at what has been achieved and how what has been done so far sets up the rest of the project for completion.

5.1 Completed Tasks

5.1.1 Identification of Simple Parallel Patterns

This is the initial requirement of the code analyser aspect of this project. Currently simple parallel patterns such as map operations have been successfully completed. This is done utilising the Clang libtooling library, using ASTMatchers and the resulting callback.

5.1.2 Inital Scaffolding of GPU Benchmarking Code

The current scaffolding of the GPU Benchmarking code provides a benchmarking class which defines the interface by which other aspects of this project must interface with the benchmarking code. Currently the interface is:

5.1.3 Basic GPU Peak Memory Performance Benchmark

The current peak memory performance benchmark is an incredibly simple memory test, the base benchmark is merely allocates memory on the device, copies memory from the host to the device, performs a no-op on the device and copies the memory back from the device to the host. This is all timed in order to calculate a peak theoretical memory bandwidth. The benchmark was written before the current version of the benchmark object existed, and as such it does not have any dynamic reponse to changing structure. It is merely a proof of concept for the peak memory performance benchmark.

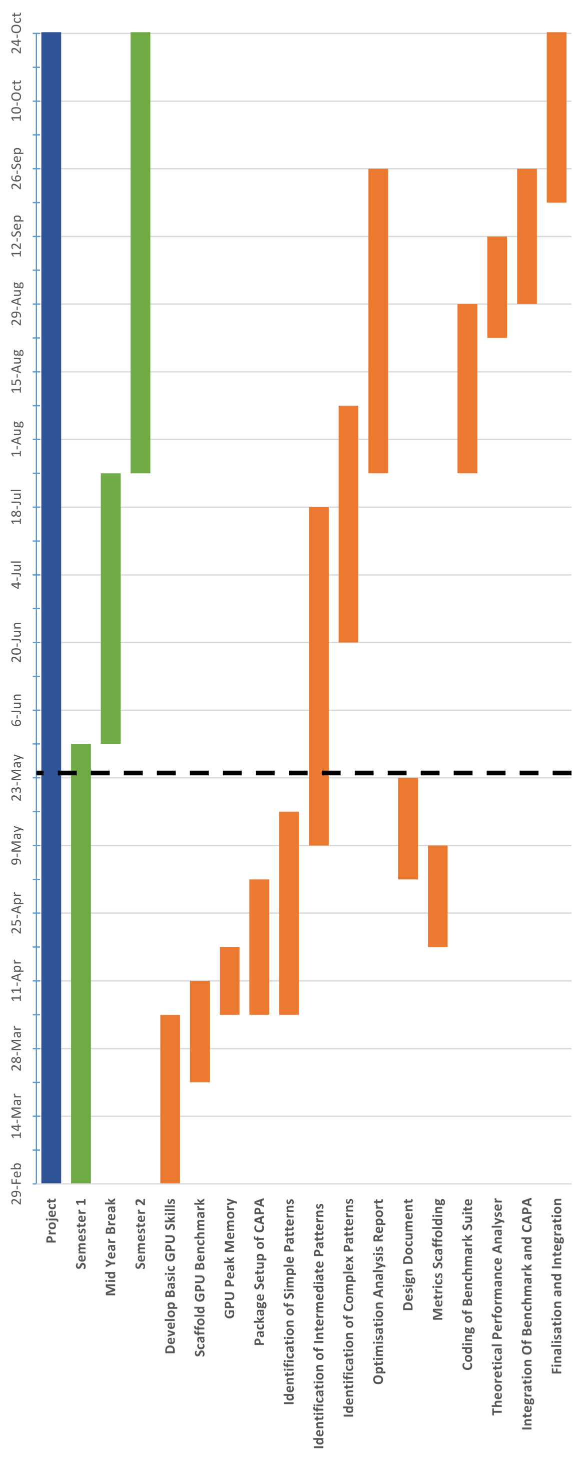
5.1.4 Scaffolding of Performance Metric Reporting

By utilising the OCLint tool, the rule and reporter interface has already been written. There are a few alterations that still need to be undertaken, as the nature of the reports is somewhat different between the original intention of OCLint, and what I am doing, however the overall scaffolding and relationship between analysis and report is finished, and quite robust.

5.1.5 Capable of Analysing Arbitrarily Large C Codebases

As the static code analyser uses the Clang libtooling, if the codebase being run on is capable of being built by clang, then it is possible for the codebase to be analysed CAPA. This is due to the heavy lifting being done by the clang compiler, and the tool hooking in only at the AST stage per compilation unit.

6 Timeline


29-Feb
14-Mar
28-Mar
11-Apr
25-Apr
9-May
23-May
6-Jun
20-Jun
4-Jul
18-Jul
1-Aug
15-Aug
29-Aug
12-Sep
26-Sep
10-Oct
24-Oct
Project
Semester 1
Mid Year Break
Semester 2
Develop Basic GPU Skills
Scaffold GPU Benchmark
GPU Peak Memory
Package Setup of CAPA
Identification of Simple Patterns
Identification of Intermediate Patterns
Identification of Complex Patterns
Optimisation Analysis Report
Design Document
Metrics Scaffolding
Coding of Benchmark Suite
Theoretical Performance Analyser
Integration Of Benchmark and CAPA
Finalisation and Integration