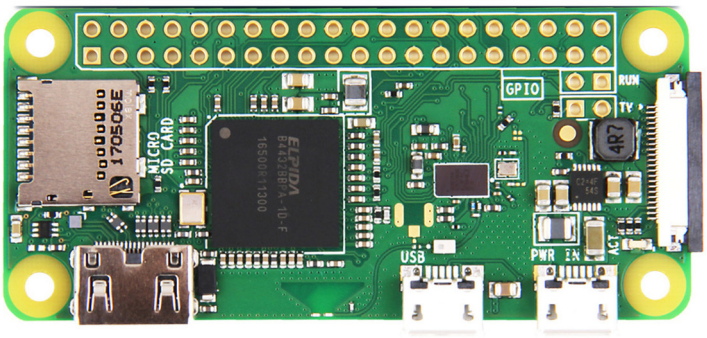
University of Cape Town



EEE3096S/EEE3095S

Embedded Systems II

Work Packages 2021

13 August 2021

Abstract

Welcome to the practicals for EEE3096S. These instruction are applicable to all practicals so please take note! It is critical to do the pre-practical/tutorial work. Tutors will not help you with questions with answers that would have been known had you done the pre-practical work. The UCT EE Wiki (wiki.ee.uct.ac.za) is a very useful point to find any additional learning resources you might need. Use the search functionality on the top right of the page.

Practical Instructions

- **Naming**
  All files submitted to Vula need to be in the format. Marks will be deducted if not.

```
pracnum_studnum1_studnum2.fileformat
```

- **Submission**
  - Work packages consist of a tutorial and a practical. Be sure you submit to the correct submission on Vula!
  - All text assignments must be submitted as pdf, and pdf only.
  - Code within PDFs should not be as a screenshot, but as text.
  - Where appropriate, each pdf should contain a link to your GitHub repository.
- **Groups**
  All practicals are to be completed in groups of 2. If a group of 2 can't be formed, permission will be needed to form a group of 3. You are to collaborate online with your partners. See more here.
- **Mark Deductions**
  Occasionally, marks will be deducted from practicals. These will be conveyed to you in the practical, but many will be consistent across practicals, such as incorrectly names submissions or including code as a screenshot instead of formatted text.
- **Late Penalties**
  Late penalties will be applied for all assignments submitted after the due date without a valid reason. They will work as 20% deduction per day, up to a maximum of 60%. After this, you will receive 0% and have the submission marked as incomplete.

Contents

Chapter 1

Work Package 2

1.1 Practical - The Unix shell and RPi Hardware

1.1.1 Overview

Pre-prac Requirements are ABSOLUTELY REQUIRED for this practical. The practical is fairly straightforward, but there is a lot of information to take in and you will likely not be able to complete it correctly if you do not understand the concepts.

This practical is designed to teach you core concepts which are fundamental to developing embedded systems in industry. It will show you the importance of C or C++ for embedded systems development. We'll start by running a program in Python - which will be our 'Golden Measure' - and comparing its execution speed to the exact same program written in C++. This will show us how using different programming languages can impact the performance of the program. From there, we'll try and improve the performance of the C code as much as possible, by using different bit widths, compiler flags and threading.

The quest for optimisation can lead one down a long, unending spiral. What is important to take away from this practical is the awareness of optimisation concepts and the ability to use good practice when benchmarking.

There's a considerable flaw in this investigation in that there is no comparison of results to our golden measure. Meaning, by the end of the practical, you will (ideally) have considerably faster execution times - but your speed-up could be entirely useless if the result you're getting is not accurate. Comparison of accuracy is left as a task for bonus marks.

1.1.2 Pre-prac Requirements

Knowledge areas

This section covers what you will need to know before starting the practical. This content will be covered in the pre-prac videos.

- Introduction to benchmarking concepts
- Introduction to cache warming and good testing methodology (multiple runs, wall clock time, speed up)
- Instruction set architecture and bit widths
- Report writing

Videos

Some videos were made to assist students with the 2019 version of this practical. They'll still be quite helpful, so here they are:

- A quick intro to compilers, flags and makefiles
- An Introduction to Benchmarking
- Report Writing

1.1.3 Outcomes

By the end of this practical you will have an appreciation for the importance of benchmarking, lower level languages, ISA and integrated hardware.

- Benchmarking
- Latex and Overleaf
- Bit widths
- Compiler Flags
- Report Writing

1.1.4 Deliverables

At the end of this practical, you must:

- Submit a report no longer than 3 (three) pages detailing your investigation. You must use IEEE Conference style. You must cite relevant literature.

1.1.5 Hardware Required

- Raspberry Pi

1.1.6 Walkthrough

Overview

1. Establish a golden measure in Python
2. Compare Python implementation to C++ implementation, in terms of accuracy and speed
3. Optimise the C++ code through parallelization and compare to golden measure
4. Optimise the C++ code through compiler flags and compare to golden measure
5. Optimise the C++ code through different bit widths, ensuring that the changes in accuracy are accounted for and compare to golden measure
6. Optimise the C++ code using a combination of parallelization, compiler flags, and bit widths and compare to golden measure

Detailed

1. Start by getting the resources off of GitHub. If you already have the local repository, you can just do a git pull. Otherwise you can clone it with:

```
$ sudo apt install git
$ git clone https://github.com/UCT-EE-OCW/EEE3096S-Pracs-2021.git --depth=1
```

   This pulls all the code from EE3096S pracs (so far) into a folder called "EEE3096S-Pracs-2021".

2. Read the README in the WorkPackage2 directory. Here is a direct link.
3. Enter into the WorkPackage2 source file directory using the `cd` command. Run the Python code to establish a golden measure. Be sure to use proper testing methodology as explained in the pre-prac content.
4. Now, run the C code (don't forget to compile it first, and every time you make a change to the source code!). This code has no optimisations in it, and also uses floats - just like the Python Implementation [1].
5. How does the execution speed compare between Python and C when using floats of 64 bits? Record your results and comment on the differences.
6. Now let's optimise through using multi-threading.
   (a) You can compile the threaded C version by running `make threaded`

[1] Floats in Python can get really weird - they don't stick at a given 32 bits. But for the sake of this practical, we're going to assume they do.

(b) The number of threads is defined in `CHeterodyning_threaded.h`

(c) Run the code for 2 threads, 4 threads, 8 threads, 16 threads and 32 threads.

(d) Does the benchmark run faster every time? Record your results, and discuss the effects of threading in your report, making note of the number of cores on the Pi 0.

7. Record your results, taking note of the most performant one. What can you infer from the results?

8. Now let's optimise through some compiler flags

   (a) Open the makefile, and in the `$(CFLAGS)` section, experiment with the following options:

Table I: Compiler Flags for optimisation

Flag	**Effect**
-O0	No optimisations, makes debugging logic easier. The default
-O1	Basic optimisations for speed and size, compiles a little slower but not much
-O2	More optimisations focused on speed
-O3	Many optimisations for speed. Compiled code may be larger than lower levels
-Ofast	Breaks a few rules to go much faster. Code might not behave as expected
-Os	Optimise for smaller compiled code size. Useful if you don't have much storage space
-Og	Optimise for debugging, with slower code
-funroll-loops	Can be added to any of the above, unrolls loops into repeated assembly in some cases to improve speed at cost of size

9. Record your results, taking note of the most performant one. Which combination of compiler flags offered the best speed up? Is it what you expected?

10. Now let's optimise using bit widths

    (a) The standard code runs using **float**

    (b) Start by finding out how many bits this is.

    (c) Run the code using 3 bit-widths: double, float, and __fp16. How do they compare

in terms of speed and accuracy? Note: for __fp16, you need to specify the flag "-mfp16-format=ieee" under your $CC flags in the makefile

11. Find the best combination of bit-width and compiler flags to give you the best possible speed up over your golden measure implementation.
12. Is there anything else you can think of that may increase performance? Perform your experiments and record your results. Bonus marks will be awarded for additional experimentation. (Hint: How can you test that the results of the Python code - our golden measure - and our fully optimised code produce the same results?)
13. Finally, record your methodology, results and conclusion in IEEE Conference format using Latex (Overleaf is recommended as an editor - See instructions here).

1.1.7 The Report

The report you need to complete needs to follow the IEEE Conference paper convention. You can find the conventions for this style online. It is recommended you use Overleaf as an editor, as it allows collaboration and handles all packages and the set up for you. In your report, you should cover the following:

- In your introduction, briefly discuss the objectives and core finding of your research
- In your methodology, discuss each experiment and how you plan to run it.
- In your results, record all your results in either tables or graphs. Be sure to include only what is relevant, but not miss out on anything that might be interesting. Remember to report on the **speedup** obtained and not just the execution speeds. Briefly justify why you got the results you did, and what each experiment did to affect the run time of your results.
- In your conclusion, mention what you did, and what your final findings are (e.g. "The program ran fastest when..."). This should only be a few sentences long.
- Look on Vula for report writing tips to avoid common mistakes and improve the readability of your report.

1.1.8 Marking

Your report will be marked according to the following:

- Following instructions
- Covering all aspects listed in this practical
- The quality of your report

Additional marks will be awarded for:

- Further experimentation
- Optimisation beyond what is described in this practical

Marks will be deducted for the following:

- Not using LaTeX
- IEEE Conference paper format not used
- Incorrectly named submission