2024

EEE3095S: COURSE PROJECT

1. OVERVIEW

The project 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 project is designed to teach you core concepts which are fundamental to developing embedded systems in industry, and it will show you the importance of C/C++ for embedded systems development.

We will begin 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 will 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 is a considerable flaw in this investigation in that there is no comparison of results to our Golden Measure — meaning that, by the end of the practical, you will (ideally) have considerably faster execution times, but your conclusions relating to speed-up could be useless if the result you are getting is not accurate. Comparison of accuracy is thus left as a task for you.

This project will make use of Qemu (Quick EMUlator); like VirtualBox, Qemu is capable of virtualization, but it can also be used for *emulation*. This is the process of creating a "guest" environment that emulates the hardware or software system inside of a different "host" system. The "guest" code is not executed directly, but instead, an interpreter interprets the "guest" code and executes equivalent operations on the "host". Emulators are commonly used to run video games whose hardware have become obsolete, such as the (*amazing*!) open-source DuckStation app used for emulation of the original PlayStation console.

2. BACKGROUND

A. Knowledge Areas

This project aims to cover:

- An introduction to benchmarking

- Instruction set architecture, compiler flags, and bit-widths
- Emulation
- Testing methodology (multiple runs, wall clock time, speed-up, etc.)
- LaTeX and Overleaf
- Report writing

B. Videos

Some videos were made to assist students with an older version of this project from a few years ago; they may still be quite helpful, so here are the links if you need them:

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

3. GETTING STARTED

1. Download and install VirtualBox. Some Windows users *may* first need to enable hardware virtualization from within the BIOS setup. Instructions and more information on how to do that can be found here.
2. Download the latest Ubuntu ISO image.
3. Get a basic understanding of git. For details regarding git and its usage, read through http://wiki.ee.uct.ac.za/Git
4. Set up VirtualBox with a virtual Ubuntu machine. Various guides are available online to assist you with this, if needed.
5. Launch the virtual machine, then enter sudo mode (you may need to enter your Ubuntu user password) and navigate into a new directory called *rpi*:
   ```
   sudo su
   mkdir rpi && cd rpi
   ```
6. Install Qemu through Terminal
   ```
   sudo apt install qemu-system
   ```
7. Clone the Qemu Raspberry Pi kernel repo:
   ```
   git clone https://github.com/dhruvvyas90/qemu-rpi-kernel.git
   ```
8. Download the latest version of Raspbian (Raspberry Pi OS):
   ```
   wget https://downloads.raspberrypi.org/raspbian_latest
   ```
   If this keeps failing, open the link in Mozilla Firefox and download it through the browser.
9. Unzip the .zip folder you just downloaded to your *rpi* folder, and rename the .img file to *rpi.img*

10. Run the following command from within the ~/rpi directory (note: The following is all one command and can be typed without the backslashes):
    ```
    qemu-system-arm -M versatilepb -cpu arm1176 -m 256 -kernel qemu-rpi-kernel/kernel-qemu-4.19.50-buster -hda rpi.img -append "dwc_otg.lpm_enable=0 root=/dev/sda2 console=tty1 rootfstype=ext4 elevator=deadline rootwait" -dtb qemu-rpi-kernel/versatile-pb-buster.dtb -no-reboot -serial stdio
    ```

    Congrats! You should now have an emulated Raspberry Pi running within an "emulated" Ubuntu (virtual machine)!

4. EXPERIMENTATION

Now it is time for you to get the actual project started! You are free to go about this however you please, but the recommended steps are summarized below:

1. Inside the emulated Pi, open a Terminal and install *git*
2. Download the project resources from GitHub:
   ```
   Git clone https://github.com/eee3096s/2024/tree/main/Project --depth=1
   ```

3. Enter the Python folder and run the Python code (using python3) to establish a Golden Measure.
4. Enter the C folder and run the C code (after compiling it, of course).
5. Optimise the C code through multi-threading. You can compile the threaded C version by running `make threaded`, and run it using `make run_threaded`.
   The number of threads is defined in *CHeterodyning_threaded.h* and can be changed accordingly for experimentation purposes.
6. Experiment with different compiler flags to let the compiler optimise the code automatically.
7. The standard code runs using *float*s; start by finding out how many bits this is, then try running the code using different bit-widths, such as *double*, *fp16*, etc.
8. Now that you have the basics under control, try different configurations! Experiment with different combinations of bit-widths, thread counts, compiler flags, and/or other parameters to give you the best possible speed-up over your Golden Measure implementation. Record and analyse your results using appropriate scientific means, and write your report based on your findings.

5. DELIVERABLES

You are required to submit a report of **up to 10 pages** (*excluding* the title page, any appendices, references, etc.) that details your investigation and results. You must use the

LaTeX/Overleaf template linked here with **IEEE-style** references to relevant literature. Being able to convey your findings clearly and concisely is an important skill, and thus you may incur penalties if you go over the page limit (or if your report is insufficient and/or poorly structured).

Your report should be well formatted and professionally presented, and the contents should generally include:

- An introduction to the report and the aim of the project
- Some background theory and information that is relevant to the project
- Design and implementation of your experiments, and your hardware and software set-up
- Results, comparisons, discussions, and analyses of your experimental outcomes
- Conclusions and recommendations for further improvements

The report will be marked out of **100**. You may work in groups of **up to four students**.