# Practical 2: The Unix shell and RPi Hardware

Bhekanani Cele, CLXBHE005
Kwena Komape, KMPKWE002

August 30, 2021

**Abstract**

In embedded systems development, engineers are concerned with both the software and hardware aspects of the system. In the implementation stage, a key decision to be made is the choice of programming language to implement the embedded software. This practical explores the significant things to consider when making such a decision. Furthermore this practical explores the concept of optimising programs execution time by using multiple threads, compiler flags, and data types. Hence finding the right combination of a number of threads, compiler flags, and data types to achieve great program speedup.

# 1 Introduction

This practical explores the core fundamental concepts of developing embedded systems used in the embedded systems industry. The main focus is to draw out the importance of choosing a programming language when developing embedded systems with limited resources. This practical will show how using different languages can impact the performance of the program. The key to take away from this practical is the awareness of the optimisation concepts and the ability to use good practise when bench-marking. When using threads to achieve speed up, using too many or too few threads is not always ideal, too many threads can lead to overhead, and using too few threads may not achieve any speed up at all. Finding the right number of threads to achieves speed up is crucial.

# 2 Methodology

## 2.1 Hardware

The hardware used is raspberry pi Zero, which has a 1GHz single-core CPU, 512MB RAM, Mini HDMI port, Micro USB OTG port, Micro USB power, HAT-compatible 40-pin header, Composite video and reset headers, and CSI camera connector (v1.3 only).

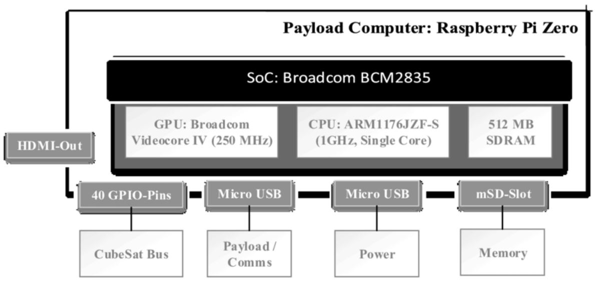


Figure 1: A Raspberry pi Zero

## 2.2 Implementation

Firstly a golden measure was established in python. This is the program used as a point of comparison, comparing its execution speed to the same program written in C/C++. This helps in showing how using different programming languages can impact the performance of a program. A quest to try and improve the performance of the C/C++ code as much as possible, by using different bit widths, compiler flags and threading is the end goal.

## 2.3 Experiment Procedure

Steps:

1. Run the Python code 10 times, record time for each run and calculate the average to establish a golden measure. See the table [3.2]

2. Compile and Run the unthreaded C code10 times, record time for each run and calculate the average. See the table [3.2]

3. Compile and Run the threaded C code for 2 threads, 4 threads, 8 threads, 16 threads, and 32 threads 10 times, record time for each run, and calculate the average. See the table [3.2]

4. Compile and Run the C code using 7 different compiler flags, record time for each run and calculate the average. See the table [3.2]

5. Compile and Run the C code using 3 bit-widths: double, float, and __fp16 data types 10 times, record time for each run and calculate the average. See the table [3.2]

# 3 Results

## 3.1 Figures

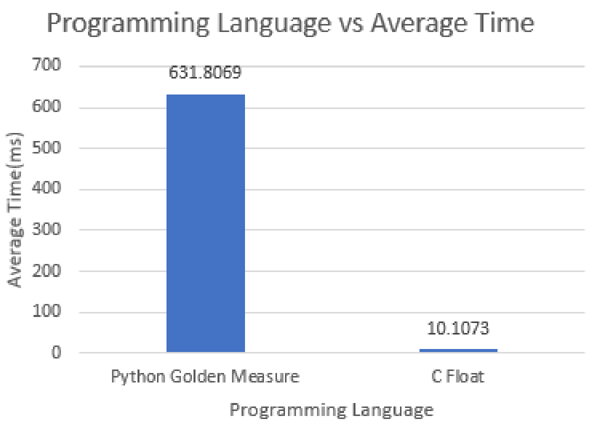


Figure 2: Program execution time of python golden measure compared to the C program

The figure 2 on the left shows how the C programming language outperforms the python program even though these are the exact same programs written in different programming languages. This is because when running the Python script, its interpreter will interpret the script line by line and generate output but in C, the compiler will first compile it and generate an output which is optimized with respect to the hardware and operating system.

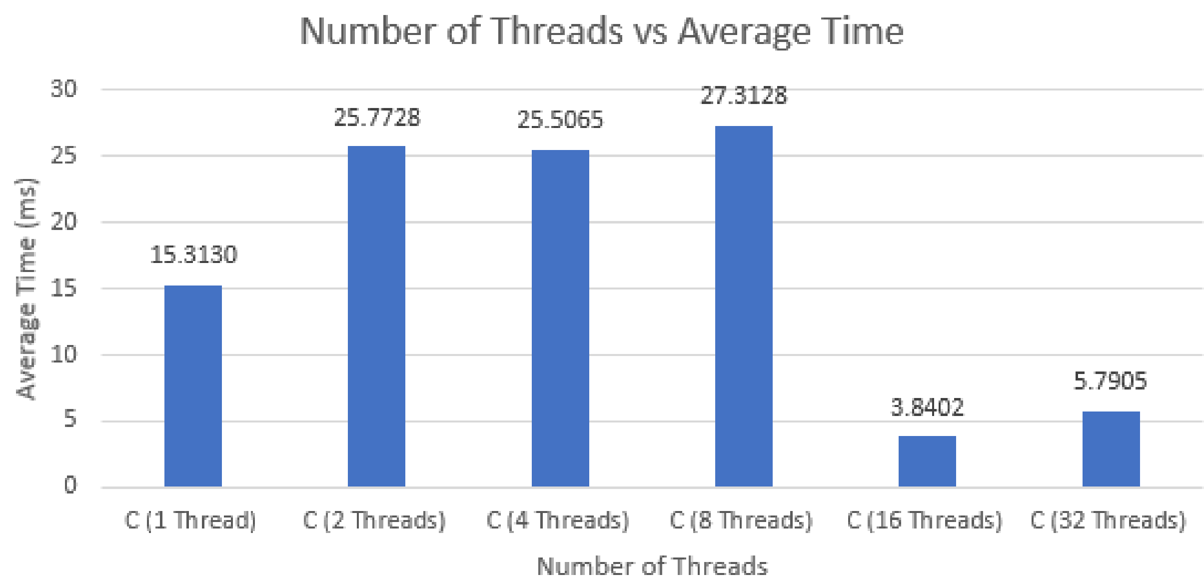


Figure 3: Program Speed at different number of threads

The above figure 3 represents how the program execution speed varies depending on how many threads the program is using. From the figure, it is clear that just increasing the number of threads a program is using does not guarantee speedup. Too many threads used can cause overhead, that is it can slow down the program execution time. Using too few threads is not ideal, as the probability of achieving speed up is very low. The key is finding just the right amount of threads that achieve the speed up. From Figure 2 the ideal number of threads to achieve a significant speedup is 8 threads. Using threads less than or greater than the ideal number of threads can impact the program speed up negatively.

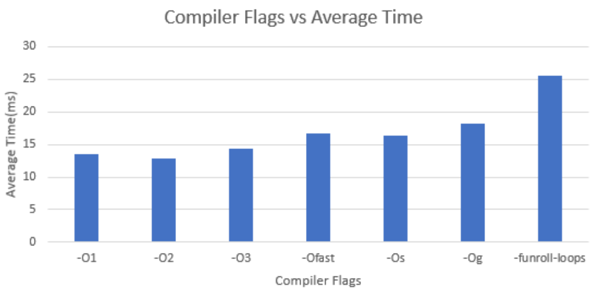


Figure 4: Program execution average Time for different compiler flags

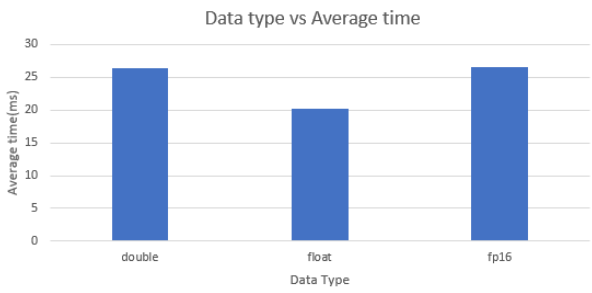


Figure 5: Average time for tens runs of each data type

The above figure 4 represents how the program execution speed varies depending on which compiler flag is used to optimise it. These flags are ranging from safe to use flags to dangerous to use. This is because some other flags compromised the program's accuracy. An example of a dangerous flag is the -Ofast flag because it breaks a few rules to go much faster. Code might not behave as expected. The -O2 flags outperform the other compiler flags.

Figure 5 above displays the average time is taken for running the program 10 times for each data type. Each program has 8 threads. From the diagram we can see that float has the lowest average time for ten runs compared to other data types (double and tp16), this shows that the float program has a high execution speed compared to others, hence faster.

## 3.2 Tables

Table 1: Comparing speed of the C program using different number of threads

| | | | C program float 64 bit (ms) | | | | | |
|---|---|---|---|---|---|---|---|---|
| Run No. | Python Float (ms) | C float 32 bits (ms) | 1 Threads | 2 Threads | 4 Threads | 8 Threads | 16 Threads | 32 Threads |
| 1 | 666.083 | 11.134 | 16.950 | 27.472 | 28.608 | 28.664 | 4.788 | 3.643 |
| 2 | 602.740 | 11.373 | 14.247 | 28.097 | 28.285 | 21.043 | 3.483 | 12.978 |
| 3 | 609.573 | 10.774 | 16.059 | 27.632 | 20.374 | 28.678 | 3.860 | 4.802 |
| 4 | 552.083 | 9.453 | 16.127 | 16.760 | 28.149 | 30.500 | 4.509 | 6.044 |
| 5 | 542.791 | 8.889 | 15.688 | 27.812 | 26.596 | 30.248 | 2.762 | 3.708 |
| 6 | 938.347 | 11.306 | 16.191 | 27.548 | 26.859 | 28.506 | 3.574 | 9.366 |
| 7 | 667.237 | 7.367 | 12.814 | 27.352 | 19.665 | 21.032 | 2.261 | 5.196 |
| 8 | 590.083 | 8.165 | 15.106 | 27.791 | 26.732 | 28.547 | 4.863 | 3.616 |
| 9 | 574.800 | 11.286 | 15.672 | 27.455 | 23.530 | 27.192 | 3.863 | 3.584 |
| 10 | 574.332 | 11.327 | 14.276 | 19.809 | 26.267 | 28.718 | 4.439 | 4.968 |
| Average | 631.8069 | 10.1073 | 15.3130 | 25.7728 | 25.5065 | 27.3128 | 3.8402 | 5.7905 |

Table 2: C program with 8 threads: Comparing different compiler flags

| | Compiler Flags | | | | | | |
|---|---|---|---|---|---|---|---|
| Run No: | -O1 | -O2 | -O3 | -Ofast | -Os | -Og | -funroll-loops |
| 1 | 13.963 | 16.709 | 13.491 | 15.879 | 15.737 | 15.657 | 25.092 |
| 2 | 16.053 | 14.374 | 16.808 | 18.023 | 18.210 | 21.391 | 21.092 |
| 3 | 12.887 | 17.999 | 16.964 | 18.143 | 15.703 | 20.023 | 21.640 |
| 4 | 10.962 | 10.113 | 19.815 | 15.813 | 17.942 | 14.029 | 29.015 |
| 5 | 12.680 | 9.114 | 13.898 | 17.607 | 13.067 | 15.750 | 25.412 |
| 6 | 11.246 | 10.508 | 11.076 | 15.964 | 14.760 | 21.325 | 29.112 |
| 7 | 19.661 | 16.707 | 14.061 | 18.205 | 13.463 | 15.573 | 28.916 |
| 8 | 13.822 | 11.968 | 11.869 | 13.485 | 18.067 | 21.338 | 25.241 |
| 9 | 9.570 | 11.698 | 12.757 | 17.817 | 18.171 | 21.350 | 28.870 |
| 10 | 13.809 | 9.958 | 13.147 | 15.612 | 17.732 | 15.566 | 21.096 |
| Average | 13.4653 | 12.9148 | 14.3886 | 16.6548 | 16.2852 | 18.2002 | 25.5486 |

Table 3: Comparing different data type.

| | Data type | | |
|---|---|---|---|
| Run No: | double | float | fp16 |
| 1 | 29.172 | 18.169 | 31.047 |
| 2 | 21.100 | 17.331 | 30.627 |
| 3 | 28.975 | 20.869 | 23.677 |
| 4 | 29.217 | 23.815 | 27.400 |
| 5 | 28.824 | 23.824 | 22.325 |
| 6 | 25.647 | 17.102 | 14.455 |
| 7 | 20.991 | 21.104 | 31.150 |
| 8 | 28.972 | 25.287 | 27.317 |
| 9 | 21.784 | 17.394 | 30.789 |
| 10 | 28.795 | 17.035 | 27.588 |
| Average | 26.348 | 20.193 | 26.638 |

# 4 Conclusion

This experiment shows that the golden measure python program is slower than the exact same program written in C/C++, which then concludes that C or C++ should be chosen over python when trying to develop embedded systems in the industry, but furthermore, the C/C++ program can still be optimized to even execute much faster, using flags, threads, and data type. The data from the experiment shows that a program with 16 threads is faster compared to other programs with different threads, the data also shows Compiler flag -O2 produces high speed, and lastly, data type float produces faster speed compared to double and fp16.