Prac 2 - Python vs. C

Matthew Terblanche[†] and Michael Wetzel[‡]
EEE3096S Class of 2019
University of Cape Town
South Africa
[†]TRBMAT002 [‡]WTZMIC001

***Abstract*—Python, C and C++ are three of the many programming languages that can be used when developing embedded systems. A program written in C can run much faster than the same program written in Python. The C code can be optimized even further by making use of different bit widths, compiler flags, threading, and particular hardware available in the ARM Processor.**

I. Introduction

Python and C are two of many programming languages used in embedded systems. The speed at which a program needs to run is a very important factor when choosing which language to use. The speed of a Python and C program can be examined using benchmark tests.

Benchmarking refers to testing a product to measure how much better or worse it is in comparison to other products [1]. The level of optimization done to a program can be measured by comparing benchmark tests of the program before and after the optimization is done. The results can be compared by looking at the speed up. Speed up is calculated using the old time divided by the new time [2]. It shows how much faster a program runs compared to another.

Using this method the decision can be made as to which programming language will have the faster execution speed and as a result which one will be best suited for the task at hand.

II. Methodology

A. Hardware

The embedded system we ran was a Raspberry Pi 3 B+. Providing power to the Raspberry pi is a AC/DC Adaptor with an output of DC 5V and 3A current.
Providing network is a 500cm UTP Ethernet cable and in the case where portable computer did not have an Ethernet port a USB 2 to Ethernet adapter was used.
To run the software needed to interact with the Pi a portable computer was used. One operating on windows and the other macOS.

B. Implementation

The first steps in Prac 2 was communication with your lab partner. The lab partners met up at the lab session, discussed how to approach this task and followed the steps laid out in the Experimental Procedure.
A decision was made to split up the tasks evenly between the two group members. Whenever an experiment was done the data was recorded in a collaborated Google spreadsheet document. Communication was done via WhatsApp and in person.

C. Experiment Procedure

First step to all the other steps is insuring the Raspberry Pi is connected to a computer and there is network. The first task is to run a specific Python program, record the execution time and as such establish a golden measure. Next the same program code was run in C language and the difference in execution time was recorded.
With all recorded times. There were 5 readings taken and the average of those calculated and that value was used.
Next the C program was optimized using various techniques. The first being multi-threading. The C program was run using a different number of threads and the execution time recorded. The program was compiled each time after its code was altered. When this task was finished the unthreaded version of the C code was used to take measurements from.
Secondly changing different types of compiler flags. Changes were made to code to include various compiler flags. The program was then recompiled, ran, and execution time recorded.
Thirdly by the changing of bit widths. Alterations were made to global.h, Prac2.c and makefile to experiment what effect different bit widths have on execution time.
Fourthly by setting different types of floating point hardware accelerator. Different commands were tried and yet again execution time was recorded.
Lastly a combination of bit widths and compiler flags were used to establish the best possible optimization for the current code.

III. Results

A. Benchmark Tests

A Python program is used as a golden measure for this practical. The initial benchmark test results are shown in table I.

TABLE I:
Initial Benchmark Test

Language	Data Type	Elapsed Time (ms)
Python	float	3904
C	float	7.228
C	double	8.823

The Python code average time is used as a golden measure in this practical to show how much of an effect running the same program using a different language can have. The golden measure is also used to show how much a program can be sped up by optimization.

The C program running with the float data type has a 540.1x speed up compared to the Python program also with a float data type. The C program running the data type of double which is 64 bits compared to the data type of float which is 32 bits run by the Python program, has a speed up of 442.5x.

B. Optimization Using Multi-Threading

The C program was run using multi-threading with a thread count of 2, 4, 8, 16, and 32. The elapsed time, speed up from the previous thread count, and speed up from the golden measure Python program are shown in table II.

TABLE II:
Multi-Threading Benchmark Test

Thread Count	Elapsed Time (ms)	Speed Up from previous	Speed Up from golden measure
2	6.279	–	621.7x
4	3.583	1.752x	1090x
8	3.593	0.997x	1087x
16	2.451	1.466x	1593x
32	2.012	1.218x	1940x

Table II shows that the benchmark tests run faster with an increase in thread count with an exception of the change from 4 threads to 8. This could be due to the Raspberry Pi having a quad core CPU and the change from single threading, one thread per core, to multi-threading. In this case 2 threads per core.

The most performant test was running the C program with a thread count of 32. This had a 1940x speed up compared to the golden measure Python program. As shown above, a C program will run faster than a Python program. The speed up of the C program compared to the Python program can be increased by making use of multi-threading and significantly increasing the thread count.

C. Optimization Using Compiler Flags

The unthreaded C program was run using different Compiler Flags. The results are shown in table III.

TABLE III:
Optimization Using Compiler Flags

Compiler flag	Elapsed Time (ms)	Speed Up from single flag	Speed Up from golden measure
-O0(none)	7.281	–	536x
-O1	3.027	–	1290x
-O2	2.995	–	1303x
-O3	3.022	–	1292x
-Ofast	3.074	–	1270x
-Os	2.992	–	1305x
-Og	5.681	–	687x
-funroll-loops + -Og	4.998	1.14x	781x
-funroll-loops + -O2	2.379	1.26x	1641x
-funroll-loops + -03	2.299	1.31x	1698x
-funroll-loops + -Os	3.051	0.98x	1279x

It seems that in general -O2 compiler flag optimizes the program the most and paired with the extra -funroll-loops flag it behaves even better. In this case different optamization compiler flags are used in different circumstances. In the case of a small C program. -O2 would be the best choice to go for fast execution time.

D. Optimization Using Bit Widths

The bit widths can be set in C code. This allows for different word sized depending on what you need. Tests were done using 16, 32 and 64 bits.

TABLE IV:
Optimization Using Bit Widths

Bid width	Accuracy	Elapsed time (ms)	Speed Up from golden measure
16	2	30.25	129x
32	4	7.227	540x
64	8	8.839	441.6x

E. Optimization Using Hardware Level Support

TABLE V:
Optimization Using Hardware Acceleration

Command used	Elapsed Time (ms)	Speed Up from golden measure
16 bit widths		
non-specified	30.28	130x
vfp3	–	–
vfpv3-fp16	7.868	496x
vfpv4	7.910	493x
neon-fp-armv8	–	–
neon-fp16	7.890	494x
vfpv3xd	30.06	130x
vfpv3xd-fp16	30.23	129x
32 bit widths		
non-specified	7.285	535x
vfp3	7.248	538
vfpv3-fp16	–	–
vfpv4	7.281	536x
neon-fp-armv8	7.296	535x
neon-fp16	–	–
vfpv3xd	7.399	527x
vfpv3xd-fp16	–	–
64 bit widths		
non-specified	8.806	443x
vfp3	8.727	447x
vfpv3-fp16	–	–
vfpv4	8.729	447x
neon-fp-armv8	8.713	448x
neon-fp16	–	–
vfpv3xd	15.90	245x
vfpv3xd-fp16	–	–

Reason for there being empty spaces: Some of the commands allow for support for 16 bit systems. In these cases using the 2 commands on a 32 bit or 64 bit system will yield no different results. The same goes for the 16 bit section for the commands who do not support 16 bits It seems that 32 bits and the -vfpv4 hardware acceleration command works the best at optimizing the program.

IV. Conclusion

A C program can run 540.1x faster than a Python program without any optimization. By optimizing using multi-threading this can be increased to 1940x faster when using 32 threads. A combination of the -O2 compiler flag and the -funroll-loops optimizes the C program the best providing great performance but the 32 thread multi-threading still provides the fastest execution time if that is the only requirement when choosing the programming language.

References

[1] "Definition - What does Benchmarking mean?" https://www.techopedia.com/definition/17053/benchmarking.

[2] J. L. Hennessy and D. A. Patterson, *Quantitative Principles of Computer Design*, 5th ed. Elsevier, 2012, pp. 46 – 47.