Emulation and Benchmarking of C Code

Kian S Frassek† and Cameron F Clark‡
EEE3096S Class of 2023
University of Cape Town
South Africa
†FRSKIA001 ‡CLRCAM007

***Abstract*—This report details the emulation and benchmarking of C code with various compiler optimisation. Python was using as our Golden Measure, and C code with 8 threads, floats, -Os and unrolled loops being the fastest**

I. Introduction

This report covers an investigation into the topics of emulation, benchmarking, and optimisation in embedded systems development. The aim is to uncover the impact of the chosen high-level language, compiler flags, bit widths, and multi-threading on a program's performance.

Emulation is a useful tool that allows the simulation of hardware and software simultaneously, and benchmarking is used to give a reference frame for all the tests. Python is chosen as our Golden Measure for these tests for its simplicity while the C code is optimised to increase the speed of the program.

Optimizations are tools used during the compiling process that are designed to improve program performance in one or more key aspects. These optimizations include changing the number of bits that the variables are stored in, adding various optimisation flags and the use of multi-threading.

II. Methodology

A. Hardware

The hardware used for this practical was a macOS laptop running Virtual Box with 4 CPU cores and 8 Gb of RAM. The Virtual Box was running Ubuntu image, which was in turn simulating a Raspberry Pi emulator. The Raspberry Pi emulator was used for the tests.

B. Implementation

The following code is the sudo code format of the two files: the python code which was used as the benchmark for the C code tests and the non-threaded C code. The third file that was used in this test was a multi-threaded version of the C code file. The 'data' and 'carrier' arrays are predefined arrays found in another tile

```
# Load data arrays from external file
Load data from external file

# Define values
c = carrier array
d = data array
result = empty array

# Main function
function main():
 Display "There are " + length of c + " samples"
 Display "using type " + type of first element in data array
 Call Timing.startlog()
 for i from 0 to length of c - 1:
 Append c[i] * d[i] to result array
 Call Timing.endlog()
```

C. Experiment Procedure

The experiment start by benchmarking in Python using the following code. All tests were run 5 times and then an average was taken:

```
cd Python
for i in {1..5}; do python3 PythonHeterodyning.py; done
```

The C code was tested next starting with the non-threaded version:

```
cd ../C
make
for i in {1..5}; do make run; done
```

The threaded version of the C code was tested next, where each test changed the defined number of threads in the *CHeterodyning_threaded.h* file. The nano command was using to edit the file:

```
nano src/CHeterodyning_threaded.h
for i in {1..5}; do make run; done
```

Where the thread count was changed in the following line to the values 1, 2, 4, 8, 16, and 32 in turn:

```
#define Thread_Count 1
```

The threaded C code was then executed for each change of thread count:

```
make threaded
for i in {1..5}; do make run_threaded; done
```

The next test involved the use of compiler optimisation flags, where one compiler optimisation flags below was added to the *makefile* at a time. The *-funroll-loops* flag was left for the next test.

III. Results

The results section is for presenting and discussing your findings. You can split it into subsections if the experiment has multiple sections or stages.

TABLE I: Optimization Flags and Effects

—c3—p—

Flag	Effect
-O0	No optimizations, makes debugging logic easier. The default
-O1	Basic optimizations for speed and size, compiles a little slower but not much
-O2	More optimizations focused on speed
-O3	Many optimizations 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	Optimize for smaller compiled code size. Useful if you don't have much storage space
-Og	Optimize 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

A. Figures

Include good quality graphs. These were produced by the Octave code presented in listings 1 and 2. You can play around with the `PaperSize` and `PaperPosition` variables to change the aspect ratio. An easy way to obtain more space on a paper is to use wide, flat figures, such as Fig.

Always remember to include axes text, units and a meaningful caption in your graphs. When typing units, a µ sign has a tail! The letter "u" is not a valid unit prefix. When typing resistor values, use the Ω symbol.

B. Tables

Tables are often a convenient means by which to specify lists of parameters. An example table is presented in table II. You can use Tablesgenerator to make your LaTeXtables.

C. Pictures and Screen-shots

When you include screen-shots, pdfLaTeX supports JPG and PNG file formats. PNG is preferred for screen-shots, as it is a loss-less format. JPG is preferred for photos, as it results in a smaller file size. It's generally a good idea to resize photos (not screen-shots) to be no more that 300 dpi, in order to reduce file size. For 2-column article format papers, this translates to a maximum width of 1024. **Never change the aspect ratio of screen-shots and pictures!**

The source used to import a picture in an exact spot, with a caption and labels

Fig. 1: An example image

```
function FormatFig(X, Y, File);
 set(gcf, 'PaperUnits'      , 'inches');
 set(gcf, 'PaperOrientation', 'landscape');
 set(gcf, 'PaperSize'       ,       [8, 4]);
 set(gcf, 'PaperPosition'   , [0, 0, 8, 4]);

 set(gca, 'FontName', 'Times New Roman');
 set(gca, 'Position', [0.1 0.2 0.85 0.75]);

 xlabel(["\n" X]);
 ylabel([Y "\n\n"]);

 setenv("GSC", "GSC"); # Eliminates stupid warning
 print(...
  [File '.pdf'],...
  '-dpdf'...
 );
end
```

Listing 1. Octave function to format a figure and save it to a high quality PDF graph

```
figure;                                     # Create a new figure
# Some code to calculate the various variables to plot...
plot(N, r, 'k', 'linewidth', 4); grid on; # Plot the data
xlim([0 360]);                              # Limit the x range
ylim([-1 1]);                               # Limit the y range
set(gca, 'xtick', [0 90 180 270 360]);      # Set the x labels

FormatFig(...                               # Call the function with:
 'Phase shift [\circ]',...                         # The x title
 'Correlation coefficient',...                     # The y title
 ['r_vs_N;_f=' num2str(f) ';_P=' num2str(P)]... # Format the file name
);
close all;                                  # Close all open figures
```

Listing 2. Example of how to use the FormatFig function

TABLE II

MY INFORMATIVE TABLE

Heading 1	Heading 2	Heading 3
Data	123	321
Data	456	654
Data	789	987

D. Maths

LaTeX has a very sophisticated maths rendering engine, as illustrated by equation 1. When talking about approximate answers, never use ± 54 V, as this implies "positive or negative 54 V". Use ≈ 54 V or ~ 54 V instead.

$$y = \int_0^\infty e^{x^2} \mathrm{d}x \tag{1}$$

IV. CONCLUSION

The conclusion should provide a summary of your findings. Many people only read the introduction and conclusion of a paper. They sometimes scan the tables and figures. If the conclusion hints at interesting findings, only then will they bother to read the whole paper.

You can also include work that you intend to do in future, such as ideas for further improvements, or to make the solution more accessible to the general user-base, etc.

Publishers often charge "overlength article charges" [1], so keep within the page limit. In EEE4084F we will simulate overlength fees by means of a mark reduction at 10% per page. Late submissions will be charged at 10% per day, or part thereof.

REFERENCES

[1] "Voluntary Page and Overlength Article Charges," http://www.ieee.org/advertisement/2012vpcopc.pdf, 2014.