CENG314 Furkan Emre YILMAZ
ASSIGNMENT 3 REPORT 230201057

## 1. Experiment Environment

Experiments ran under below specification Linux installed notebook computer:

CPU: Intel© Core™ i5-5200U CPU @ 2.20GHz × 2

RAM: 12GB DDR3

Linux OS: Linux Mint 19 Cinnamon

Linux Kernel: 4.15.0-20-generic

GCC Version: gcc (Ubuntu 7.3.0-27ubuntu1~18.04) 7.3.0

All the section of the experiment fully automated by shell script. When experiments taking place there is no interaction with the computer. Also internet connection of the computer disabled to maximize the performance.

## 2. Purpose

In these set of experiments we are trying to understand what can effect our execution time and executable size when we compile our programs. We are trying different methods to see the difference between each configuration. We both try to take the power of gcc compiler optimization and manual improvements to maximize the performance of the program compiled.

## 3. Expectation

Before I start to experiment, I love to put my bet on the table. In this set of experiment based on my gained experience in CENG314 lectures I have tried to guess the results. My guess is like that

**First Part:** I think that gcc compiler optimization will decrease the execution time nearly %10 – 30 depending on the optimization level applied. However I believe when we continue to increase the optimization level our gain will decrease some amount. For the executable size I expect to observe %5 - %10 increase.

**Second Part:** I think that with each step applied our execution time decrease some amount my assumption nearly %5 - %30. Except for the last one which we apply unrolling. Unrolling may have different behavior from other optimization techniques. For the O3 optimization applied I always expect better execution times. For the executable size I expect %1 - %2 increase for the gcc O3 optimization not applied versions.

## 4. Data Collection

I fully automate the experiment by writing shell script thanks to the shell script I just ran, then I go the take my coffee and have fun.

Shell script output written into the two different directories “first_part” and “second_part” these directories includes all the executable files used in experiment plus all the execution results written into the “results.txt”.

CENG314 Furkan Emre YILMAZ
ASSIGNMENT 3 REPORT 230201057

**First Part Results Obtained From “first_part/results.txt” file:**

| Optimization Level | Executable Size | Execution Time(avg of 10) |
|---|---|---|
| -O0 | 8.504 bytes | 136.583253 sec |
| -O1 | 8.504 bytes | 46.33007 sec |
| -O2 | 8.504 bytes | 35.03274 sec |
| -O3 | 8.504 bytes | 35.16761 sec |
| -Ofast | 9.960 bytes | 34.85788 sec |

**Second Part Results Obtained From “second_part/results.txt” file:**

| Optimized Version | Executable Size | Execution Time(avg of 10) |
|---|---|---|
| unoptimized | 8.504 bytes | 137.01723 sec |
| unoptimized_o3 | 8.504 bytes | 35.01389 sec |
| no_fc (Function call eliminated) | 8.480 bytes | 114.95216 sec |
| no_fc_o3 | 8.488 bytes | 35.02843 sec |
| no_fc_temp (Function call eliminated + accumulated in temporary) | 8.480 bytes | 84.81521 sec |
| no_fc_temp_o3 | 8.488 bytes | 31.89568 sec |
| no_fc_temp_unrolled (Function call eliminated + accumulated in temporary + loop unrolled) | 8.496 bytes | 24.85383 sec |
| no_fc_temp_unrolled_o3 | 8.496 bytes | 7.71464 sec |

## 5. Discussion

Lets talk about what we collected. First of all I want to admit that when I started to take the first results of the experiments. I shocked because improvement I think is huge. Only changing the gcc optimization flag we can double our performance. It is hard to believe but I guess it is the reality but the really interesting one.

For the first part results what I can say is that:

The -O1 optimization flag makes the biggest improvement in our performance when we look the gcc description we can see that different kind of techniques used to tune the performance and they are seems to working. After the -O1 flag performance increase not as high as the first one.

When we look at the executable size we see not any difference except for the -Ofast flag -Ofast includes some inlining techniques which can cause the increase of the executable size.

For the second part results what I can say is that:

Inlining the functions increase our performance because we are not need to jump somewhere to execute or we don’t need the method call overhead stack allocation and other kind of stuff.

Adding temporary variable on top of inlining which enables to use register of the system increase our performance some amount and I think it is logical because making operation in the register more faster than the making operation in main memory(RAM)

After all adding loop unrolling has really different affect. I again shocked when I saw the improvement. I know when we unroll twice because of 2000 % 3 != 0 we decrease our computation labor. However the things happening in loop unrolling is really fascinating. Based on my experience in CENG314 lectures I think when we unroll the loop computers can paralize the job well and use pipelining effectively to tune the performance as much as possible. I will discuss these result with my instructor and friends it is really hard to believe.

For the executable size I think there is no considerable difference.

## 6. Summary

Now we are in the end, from these experiment I think we really see how we can tune our executable programs performance by considering some little but important things. The things I learned from these experiments are really hard to forgot after that point when I writing the code I will probably consider different things to tune my performance when I need more performance out of my existing program.