Chih Han “Josh” Yeh
Lab 0
ENGEC 527

Part 1.

1a.

- Intel Core i7-9700
- Operating Frequency: 3000 MHz or 3.0 GHz
- 8 Cores

1b.

- 3 levels of cache (Up to L3): a 64KB L1 Cache per core, 256KB L2 Cache per core, and a shared 12MB L3 Cache
- Coffee Lake
- 8 real physical cores, logical cores = real cores x threads that can run on each core
- Max Memory Bandwidth = 41.6 GB/s, DDR4-2666, Maximum 128GB
- Base Frequency: 3.00 GHz, Turbo Frequency: 4.70 GHz

Part 2

2a.

- Accuracy is determined by comparing the measured value with an accepted control/standard value; calculating how far or close the measurement is from the acceptable standard.
- Without using any gadgets or machines, the only way I could determine accuracy is to know the current time or the time that has passed since a certain time (the known measured standard) and begin counting/timing while accounting for possible human error.
- The RDTSC method seems to be the most accurate and/or precise method out of the three as it is still generating and displaying the time it took to run through the final 3 to 4 loops.

2b.

- The RDTSC method is tied to the CPU’s clock frequency, but only since it has been booted. The issue with this method would be the difference in clocking frequencies if it were run on a system using a multi-core CPU, where certain features may lead to different clock frequencies for each core, so the timing would be asynchronous.

2c.

- gettimeofday() would require a change in the GET_TOD_TICS (converts microseconds to seconds) but this constant is default to the system so it would remain unchanged.
- For the RDTSC method a change would occur in the CLK_RATE constant from 2.0e9 to 3.0e9 because the base frequency of the CPU is 3.00 GHz.

- For the times() method a change would occur in the GET_SECONDS_TICS constant if the number of clock ticks per second is different than 100 in the system, but that is untrue so it remains unchanged.
- Conclusion: Change the CLK_RATE constant value to 3.0e9

2d.
The resolution of the measurements go to the nanosecond.

Measurements for Standard Deviation

Attempts/Runs of test_clock_gettime.c	Time Taken (s)
1	1.037911529
2	1.026025398
3	1.032055827
4	1.030483051
5	1.053938373

Arithmetic Average = 1.036082836
Sum of Squared Difference = 0.000471
Variance = 9.41782×10^{-5}
Standard Deviation = 0.009704543 seconds

Part 4
4b.

- Since noise is more detectable in smaller and specific data sets, increasing the range of sizes of the data points would be a viable method to eliminate the noise and create linear-appearing data. Another viable way is to use the aforementioned CPE, which utilizes slopes and regression methods that would eliminate noise as much as possible.

4c.

- The estimated CPE using the slope of the number of cycles plotted against time (output of the to psum method arrays multiplied by the CPU clock frequency 3.0 GHz), is 5.82 and 0.416 for psum1 and psum2 respectively. The figures are comparatively lower than the examples from the textbook; as a speculation one of the reasons this occurs is because the CPU clock frequency of the system is significantly higher than the system that the example provided by the textbook is measuring the CPE.

Part 5.
5a.

- What got printed is the line “Starting a loop” and the word “done” after compiling and running the program. The time command displays the elapsed, user, and system time for

the program's execution with O0 optimization. Real and User time are very similar (0.584s vs 0.582s) whereas Sys time is significantly faster (0.001s).

5b.

- Same as part a, the words "Starting a loop" and "done" were printed after execution. But this time the execution times were significantly faster (real: 0.048s, user: 0.045s, sys: 0.002s).

5d.

- The file is significantly shorter, displaying fewer instructions than the O0 optimization method. The loop in .L3 is removed, and the counter starts with 200000001 instead of 200000000. Most variables and calculations that are unused are removed, and the loop itself has been simplified as well. The resulting assembly code length is almost half as long compared to the O0 optimization method.

5e.

- The time it took to execute the program is almost if not exactly the same as the O0 optimization method. Because quasi_random is needed after the loop so it requires the compiler to preserve the loop similarly to O0 optimization. Therefore most of the already eliminated variables are retained, causing the compilation time to almost be a 1:1 exact copy of the O0 optimization.

Part 6.

6c.

- The memory bandwidth from executing stream.c is 11367.066s MB/s which translates to 11.1367 GB/s, approximately a quarter of the maximum memory bandwidth of the provided specifications from Intel.

6f.

- Just like the Roofline Model, when arithmetic intensity and GFLOPs/s would still display a linear relationship when plotted against each other, the machine's performance during a program's execution is bound by the processor's peak memory bandwidth. In the case of our systems, this occurs when the AI ranges from 0.125 (⅛) all the way to 4.0. Arithmetic Intensity above 4.0 seems to be the limit at which the program execution becomes compute-bound, meaning the rate of GFLOPs's increase is not as high as the memory-bound range due to the computational power of the CPU.
- Attainable GFLOPs/s from 0.125 AI to 4.0 AI are all within expectation or even exceeded it (theoretical attainable AI at 0.125 = 0.581, actually measured = 2.212, theoretical attainable AI at 4.0 = 18.59, actually measured = 19.09). The slope of the memory bound region is around 4.68, which is much lower than the 41 of the maximum memory bandwidth as provided by Intel's specification. The reason why it is significantly lower could be due to other system bottlenecks, system noise, or unoptimized program execution.

Part 7.

7a.

- I would say I do, I am very unfamiliar with C because I came from a non-engineering background and my programming is limited to Python, therefore reading C and assembly language was quite difficult

7b.

- It took me over 7 hours to complete the assignment, from repeatedly going through the environment, reading through the code, and figuring out how to work around LibreOffice if my data selection or types were incorrect when generating the trendline and plots.

7c.

- I do not observe any sections of the assignment to be unreasonably time consuming. More so it would make it simpler if the lab file would provide more context and information on some of the things the questions were asking i.e parts 4 and 6 with regards to the commenting about the attainable GFLOPs/s and also methods of reducing noise. Otherwise there weren't necessarily any problems with the lab, more so issues with handling the computer lab systems because some applications or processes needed to be re-ran when using remote hosting ssh or even in person (either an OS unoptimized issue or hardware issue)