In this problem, you are to select a set of 3 single-threaded benchmark programs. Try to provide some diversity in the set of workloads you have chosen (e.g., floating point, integer, memory intensive, sparse). Then complete the folowing experients using these benchmarks. Make sure to provide as many details as possible about the systems you are using, and where you obtained the source code for these benchmarks.

a. Compile and run these 3 benchmarks on a Linux-based system of your choosing (you can also use either the COE systmes or the Discovery systems). Provide detailed information about the platform you chose, including the model and frequency of the CPU, number of cores, the memory size, and the operating system version. You should record the execution time, averaged over 10 runs of the program. Is any run of the program faster than another? If so, comment on any difference you observe in your write-up, providing some justification for the differences.
b. Next, explore the compiler optimizations available with the compiler on your system (e.g., gcc or g++), and report on the performance improvements found for the 3 workloads. Describe the optimization you applied, and provide insight why each of your benchmarks benefitted from the specific compiler optimization applied.
c. Assume that you were going to rewrite these applications using pthreads. Descirbe how you would use pthreads to obtain additional speedup by running your benchmark on multiple cores.

*Answer to this question should be included in your homework 1 write-up in pdf format.

System Enviornment (Discorvery Cluster)
Model: 16 Cores Intel(R) Xeon(R) CPU E5-2650 0 @ 2.00GHz
Memory Size: 131812036 kB
OS Version: 3.10.0-957.21.3.el7.x86_64

1) CPU Benchmark – 1,000,000,0 Loop
(https://github.com/arihant15/Performance-Evaluation-Benchmark)
① 3.876634 G-FLOPs
② 4.741265 G-FLOPs
③ 5.355657 G-FLOPs
④ 4.835216 G-FLOPs
⑤ 3.209266 G-FLOPs
⑥ 3.780592 G-FLOPs
⑦ 3.761819 G-FLOPs

⑧ 2.972109 G-FLOPs
⑨ 2.881707 G-FLOPs
⑩ 2.928540 G-FLOPs
Average: 3.834281 G-FLOPs

The performance using O flag increased to 13.180908 G-FLOPs
The performance using O2 flag increased to 2629.478705 G-FLOPs
The performance using O3 flag increased to 3281.607450 G-FLOPs

2) Memory Benchmark – 1048576 Bytes
(https://github.com/arihant15/Performance-Evaluation-Benchmark)
① 230 ns
② 231 ns
③ 221 ns
④ 252 ns
⑤ 219 ns
⑥ 204 ns
⑦ 219 ns
⑧ 208 ns
⑨ 223 ns
⑩ 230 ns
Average: 223.7 ns

The latency using O flag is 221 ns
The latency using O2 flag is 217 ns
The latency using O3 flag is 213 ns

3) Floating Point Benchmark
(https://github.com/R3tr074/oh_my_cpu)
① avx fp64 5.9261 G-FLOPs
② avx fp64 6.0691 G-FLOPs
③ avx fp64 5.8591 G-FLOPs
④ avx fp64 5.9269 G-FLOPs
⑤ avx fp64 6.0471 G-FLOPs
⑥ avx fp64 5.9200 G-FLOPS
⑦ avx fp64 6.2127 G-FLOPs
⑧ avx fp64 6.2834 G-FLOPs
⑨ avx fp64 5.8562 G-FLOPs

⑩ avx fp64 5.6694 G-FLOPs
Average: ave fp64 5.9770 G-FLOPs

As the source makefile code is already O3 optimised, I have dropped to O2 optimised for comparison.
The performance using O2 flag decreased to avx fp64 5.7768 G-FLOPs
The performance using O flag decreased to avx fp64 5.6721 G-FLOPs

O flag optimisation aims to reduce the size of the code and the speed of the executable code as much as possible without affecting the speed of compilation, using a number of optimisation algorithms. O2 flag sacrifices some of the compilation speed and uses almost all the optimisation algorithms supported by the target configuration to improve the speed of the target code, in addition to performing all the optimisations performed by O flag. O3 flag, in addition to performing all the optimization options of O2, generally takes vectorization algorithms to increase the degree of parallel execution of the code, using pipelines, Cache, etc. in modern CPUs. if I use pthreads to optimize benchmark, I will split the operations functions out to individual threads to take up more resources per unit of time.