Programming Assignment #1

1. Hardware

The machine used for compilation, execution, and benchmarking was the 2021 Macbook Pro with the Apple M1 pro chip.

- 8-core CPU with 6 performance cores and 2 efficiency cores
- 14-core GPU
- 16-core Neural Engine
- 200GB/s max memory bandwidth

2. Compilation:

The gcc provided by apple did not support openmp so I had to install another version with homebrew. The variable OMP_NUM_THREADS could be set to vary the number of threads used.

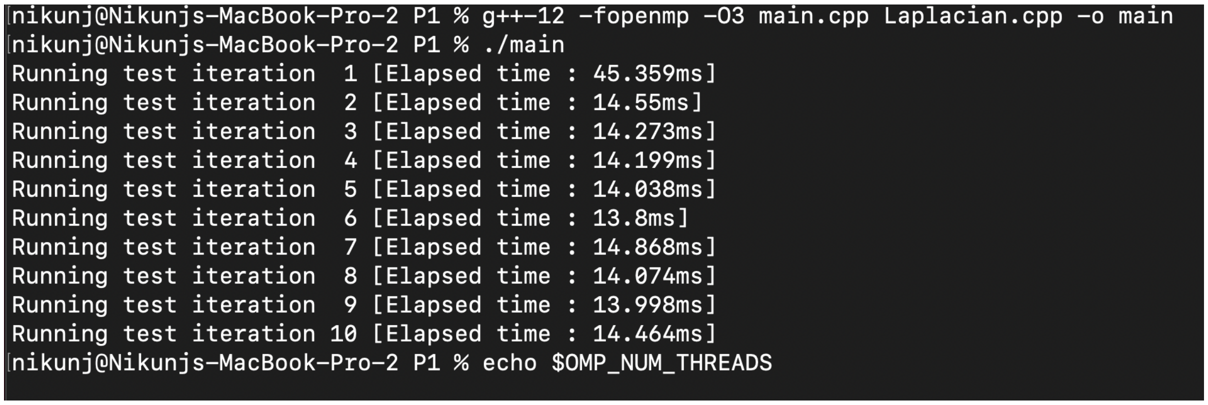
Commands used for compilation and execution:

```
g++-12 -fopenmp -O3 main.cpp Laplacian.cpp -o main
./main
```

3. Performance Evaluation

Varying the number of threads:

Without setting the OMP_NUM_THREADS variable and i→j→k loop ordering:

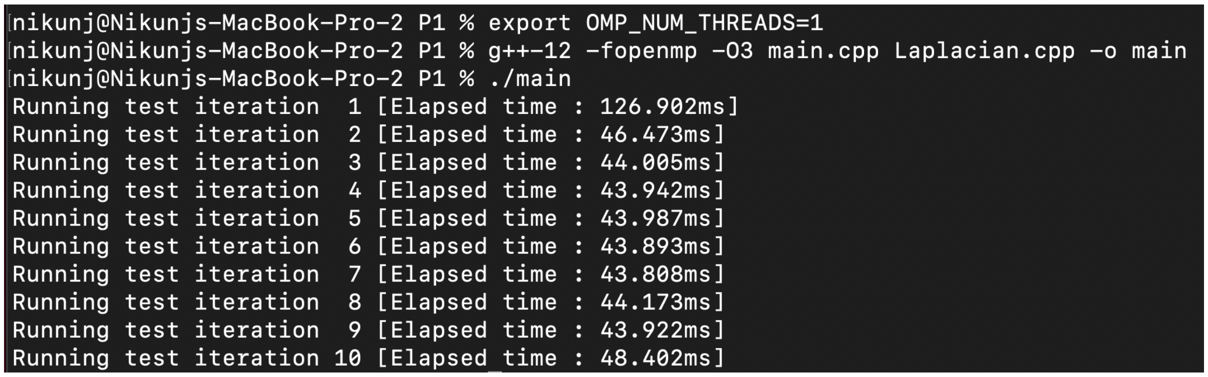

```
nikunj@Nikunjs-MacBook-Pro-2 P1 % g++-12 -fopenmp -O3 main.cpp Laplacian.cpp -o main
nikunj@Nikunjs-MacBook-Pro-2 P1 % ./main
Running test iteration  1 [Elapsed time : 45.359ms]
Running test iteration  2 [Elapsed time : 14.55ms]
Running test iteration  3 [Elapsed time : 14.273ms]
Running test iteration  4 [Elapsed time : 14.199ms]
Running test iteration  5 [Elapsed time : 14.038ms]
Running test iteration  6 [Elapsed time : 13.8ms]
Running test iteration  7 [Elapsed time : 14.868ms]
Running test iteration  8 [Elapsed time : 14.074ms]
Running test iteration  9 [Elapsed time : 13.998ms]
Running test iteration 10 [Elapsed time : 14.464ms]
nikunj@Nikunjs-MacBook-Pro-2 P1 % echo $OMP_NUM_THREADS
```

Best Runtime: 13.8 ms

Bandwidth: 77.8 GB/s

Effective Bandwidth: 38.9%

With 1 Thread and i→j→k loop ordering:

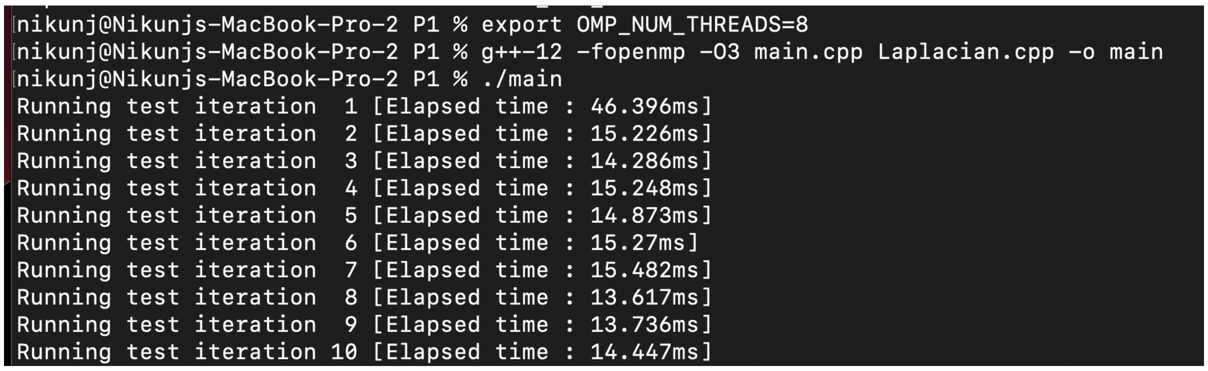

```
nikunj@Nikunjs-MacBook-Pro-2 P1 % export OMP_NUM_THREADS=1
nikunj@Nikunjs-MacBook-Pro-2 P1 % g++-12 -fopenmp -O3 main.cpp Laplacian.cpp -o main
nikunj@Nikunjs-MacBook-Pro-2 P1 % ./main
Running test iteration  1 [Elapsed time : 126.902ms]
Running test iteration  2 [Elapsed time : 46.473ms]
Running test iteration  3 [Elapsed time : 44.005ms]
Running test iteration  4 [Elapsed time : 43.942ms]
Running test iteration  5 [Elapsed time : 43.987ms]
Running test iteration  6 [Elapsed time : 43.893ms]
Running test iteration  7 [Elapsed time : 43.808ms]
Running test iteration  8 [Elapsed time : 44.173ms]
Running test iteration  9 [Elapsed time : 43.922ms]
Running test iteration 10 [Elapsed time : 48.402ms]
```

Best Runtime: 43.2 ms

Bandwidth: 24.9 GB/s

Effective Bandwidth: 12.5%

With 8 Threads and ijk loop ordering:



```
nikunj@Nikunjs-MacBook-Pro-2 P1 % export OMP_NUM_THREADS=8
nikunj@Nikunjs-MacBook-Pro-2 P1 % g++-12 -fopenmp -O3 main.cpp Laplacian.cpp -o main
nikunj@Nikunjs-MacBook-Pro-2 P1 % ./main
Running test iteration  1 [Elapsed time : 46.396ms]
Running test iteration  2 [Elapsed time : 15.226ms]
Running test iteration  3 [Elapsed time : 14.286ms]
Running test iteration  4 [Elapsed time : 15.248ms]
Running test iteration  5 [Elapsed time : 14.873ms]
Running test iteration  6 [Elapsed time : 15.27ms]
Running test iteration  7 [Elapsed time : 15.482ms]
Running test iteration  8 [Elapsed time : 13.617ms]
Running test iteration  9 [Elapsed time : 13.736ms]
Running test iteration 10 [Elapsed time : 14.447ms]
```

Best Runtime: 13.6 ms

Bandwidth: 78.9 GB/s

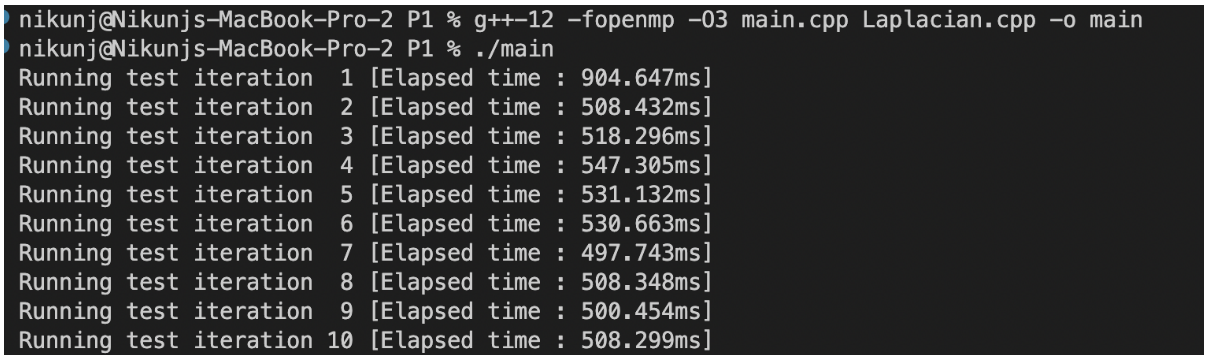
Effective Bandwidth: 39.5%

Varying the number of threads signficantly changes the runtime as expected. Getting around 40% of the peak performance is expected in this case because the native gcc compiler cannot be used for openmp and a certain amount of emulation needs to be done. Considering this, 40% is good for the effective bandwidth.

Changing the ordering of the for loops:

The OMP_NUM_THREADS variable was not set for this part so openmp will use all the available cores on the machine. This setting gave great performance in the earlier section

k→j→i ordering:



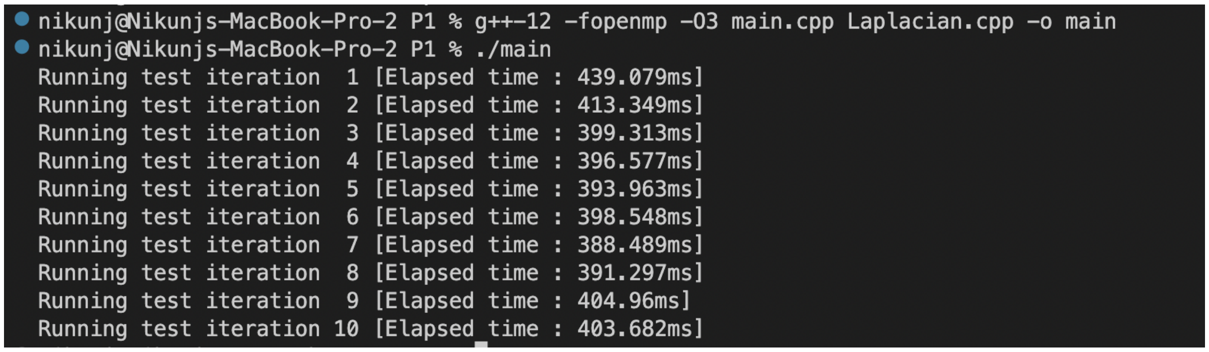
```
nikunj@Nikunjs-MacBook-Pro-2 P1 % g++-12 -fopenmp -O3 main.cpp Laplacian.cpp -o main
nikunj@Nikunjs-MacBook-Pro-2 P1 % ./main
Running test iteration  1 [Elapsed time : 904.647ms]
Running test iteration  2 [Elapsed time : 508.432ms]
Running test iteration  3 [Elapsed time : 518.296ms]
Running test iteration  4 [Elapsed time : 547.305ms]
Running test iteration  5 [Elapsed time : 531.132ms]
Running test iteration  6 [Elapsed time : 530.663ms]
Running test iteration  7 [Elapsed time : 497.743ms]
Running test iteration  8 [Elapsed time : 508.348ms]
Running test iteration  9 [Elapsed time : 500.454ms]
Running test iteration 10 [Elapsed time : 508.299ms]
```

Best Runtime: 497 ms

Bandwidth: 2.2 GB/s

Effective Bandwidth: 1.1%

i→k→j ordering:



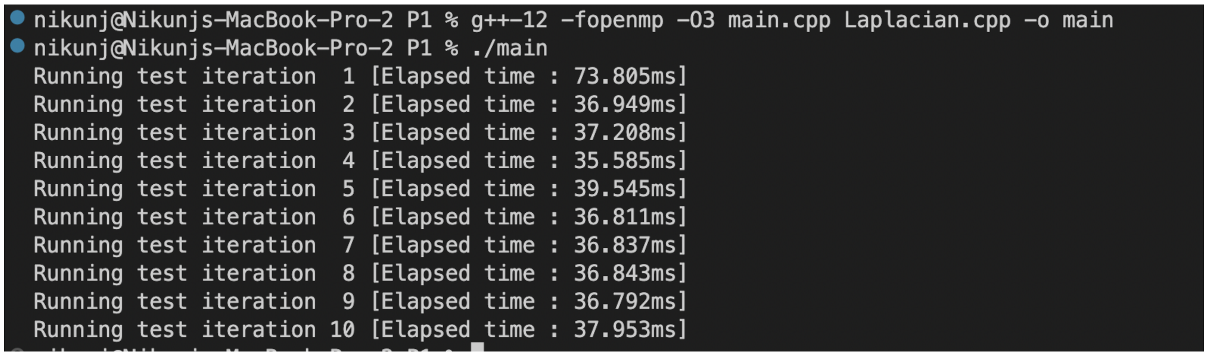
```
nikunj@Nikunjs-MacBook-Pro-2 P1 % g++-12 -fopenmp -O3 main.cpp Laplacian.cpp -o main
nikunj@Nikunjs-MacBook-Pro-2 P1 % ./main
Running test iteration  1 [Elapsed time : 439.079ms]
Running test iteration  2 [Elapsed time : 413.349ms]
Running test iteration  3 [Elapsed time : 399.313ms]
Running test iteration  4 [Elapsed time : 396.577ms]
Running test iteration  5 [Elapsed time : 393.963ms]
Running test iteration  6 [Elapsed time : 398.548ms]
Running test iteration  7 [Elapsed time : 388.489ms]
Running test iteration  8 [Elapsed time : 391.297ms]
Running test iteration  9 [Elapsed time : 404.96ms]
Running test iteration 10 [Elapsed time : 403.682ms]
```

Best Runtime: 388 ms

Bandwidth: 2.7 GB/s

Effective Bandwidth: 1.4%

j→i→k ordering:



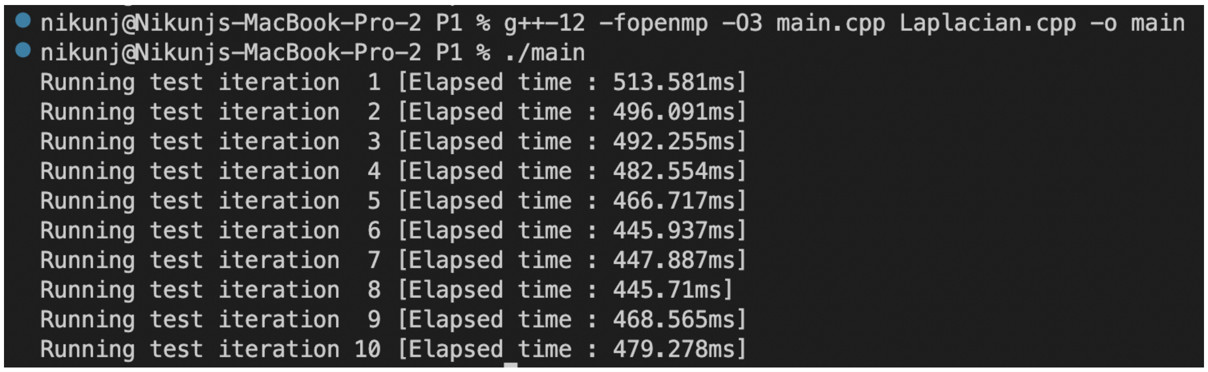
```
nikunj@Nikunjs-MacBook-Pro-2 P1 % g++-12 -fopenmp -O3 main.cpp Laplacian.cpp -o main
nikunj@Nikunjs-MacBook-Pro-2 P1 % ./main
Running test iteration  1 [Elapsed time : 73.805ms]
Running test iteration  2 [Elapsed time : 36.949ms]
Running test iteration  3 [Elapsed time : 37.208ms]
Running test iteration  4 [Elapsed time : 35.585ms]
Running test iteration  5 [Elapsed time : 39.545ms]
Running test iteration  6 [Elapsed time : 36.811ms]
Running test iteration  7 [Elapsed time : 36.837ms]
Running test iteration  8 [Elapsed time : 36.843ms]
Running test iteration  9 [Elapsed time : 36.792ms]
Running test iteration 10 [Elapsed time : 37.953ms]
```

Best Runtime: 35.5 ms

Bandwidth: 30.2 GB/s

Effective Bandwidth: 15.2%

j→k→i ordering:

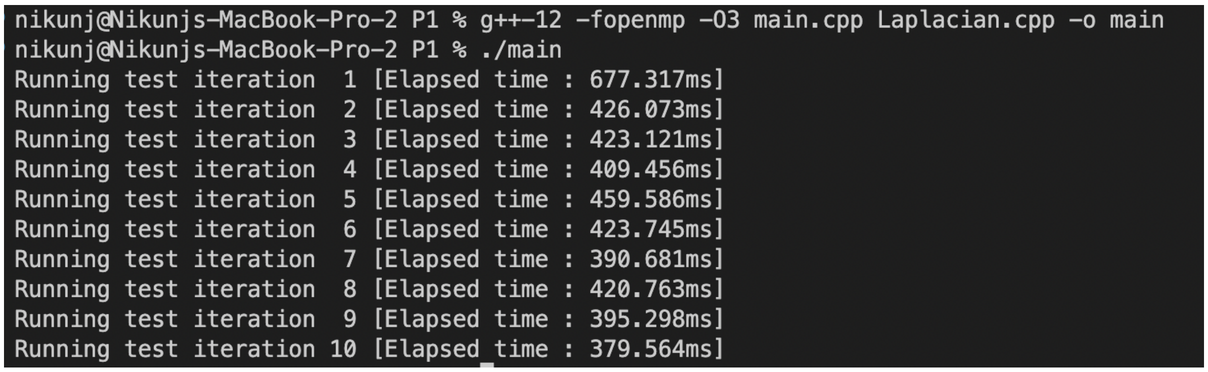

```
nikunj@Nikunjs-MacBook-Pro-2 P1 % g++-12 -fopenmp -O3 main.cpp Laplacian.cpp -o main
nikunj@Nikunjs-MacBook-Pro-2 P1 % ./main
Running test iteration  1 [Elapsed time : 513.581ms]
Running test iteration  2 [Elapsed time : 496.091ms]
Running test iteration  3 [Elapsed time : 492.255ms]
Running test iteration  4 [Elapsed time : 482.554ms]
Running test iteration  5 [Elapsed time : 466.717ms]
Running test iteration  6 [Elapsed time : 445.937ms]
Running test iteration  7 [Elapsed time : 447.887ms]
Running test iteration  8 [Elapsed time : 445.71ms]
Running test iteration  9 [Elapsed time : 468.565ms]
Running test iteration 10 [Elapsed time : 479.278ms]
```

Best Runtime: 445 ms

Bandwidth: 2.4 GB/s

Effective Bandwidth: 1.2%

k→i→j ordering:


```
nikunj@Nikunjs-MacBook-Pro-2 P1 % g++-12 -fopenmp -O3 main.cpp Laplacian.cpp -o main
nikunj@Nikunjs-MacBook-Pro-2 P1 % ./main
Running test iteration  1 [Elapsed time : 677.317ms]
Running test iteration  2 [Elapsed time : 426.073ms]
Running test iteration  3 [Elapsed time : 423.121ms]
Running test iteration  4 [Elapsed time : 409.456ms]
Running test iteration  5 [Elapsed time : 459.586ms]
Running test iteration  6 [Elapsed time : 423.745ms]
Running test iteration  7 [Elapsed time : 390.681ms]
Running test iteration  8 [Elapsed time : 420.763ms]
Running test iteration  9 [Elapsed time : 395.298ms]
Running test iteration 10 [Elapsed time : 379.564ms]
```

Best Runtime: 379 ms

Bandwidth: 2.8 GB/s

Effective Bandwidth: 1.4%

Brief Commentary:

The significantly worse performance for the k→j→i ordering makes sense because the way the elements are arranged results in a lot of cache misses. The way the matrix elements are stored in memory is very different to this ordering. The $i \rightarrow k \rightarrow j$ ordering did have worse performance but not as bad as the $k \rightarrow j \rightarrow i$ ordering. This could be because the first loop was in line with the arrangement of elements in memory and the impact on performance was not as severe. The base implementation was still an order of magnitude faster than when the loop ordering was changed. This shows the importance of caching for performance.