Student Name: Rakyan Adhikara

Student ID: 219548135

Task M2.T1P (Parallel Matrix Multiplication)

Codes are available on attached files.

Time taken for:

1. Sequential:
   a. Size 10 x 10:

   ```
   $ ./matrixMultiply
   Time taken by function: 5 microseconds
   ```

   b. Size 100 x 100:

   ```
   $ ./matrixMultiply
   Time taken by function: 3491 microseconds
   ```

   c. Size 1000 x 1000:

   ```
   $ ./matrixMultiply
   Time taken by function: 6903461 microseconds
   ```

2. Pthread:
   a. Size 10 x 10:

   ```
   $ ./matrixMultiply_pthread.exe
   Time taken by function: 743 microseconds
   ```

   b. Size 100 x 100:

   ```
   $ ./matrixMultiply_pthread.exe
   Time taken by function: 3867 microseconds
   ```

   c. Size 1000 x 1000:

   ```
   $ ./matrixMultiply_pthread.exe
   Time taken by function: 5665159 microseconds
   ```

3. OpenMP:
   a. Size 10 x 10:

   ```
   $ ./matrixMultiply_openmp.exe
   Time taken by function: 618 microseconds
   ```

   b. Size 100 x 100:

   ```
   $ ./matrixMultiply_openmp.exe
   Time taken by function: 1555 microseconds
   ```

c. Size 1000 x 1000:

```
$ ./matrixMultiply_openmp.exe
Time taken by function: 1487544 microseconds
```

Based on my findings:

1. Using parallel programming does improve performance on matrix multiplication, which is how it must be compared with sequential programming. However, OpenMP improves the performance significantly compared with pthread implementation.
2. Size of the matrices also affects the performance. In size 10 x 10, sequential has the lowest execution time. In both size 100 x 100 and 1000 x 1000, OpenMP has the lowest execution time. In addition, surprisingly, in size 100 x 100, pthread has higher execution time compared with sequential, which makes it the least efficient. Pthread also has the highest execution time in size 10 x 10.