Student Name: Rakyan Adhikara

Student ID: 219548135

Task M2.T1P (Parallel Matrix Multiplication)

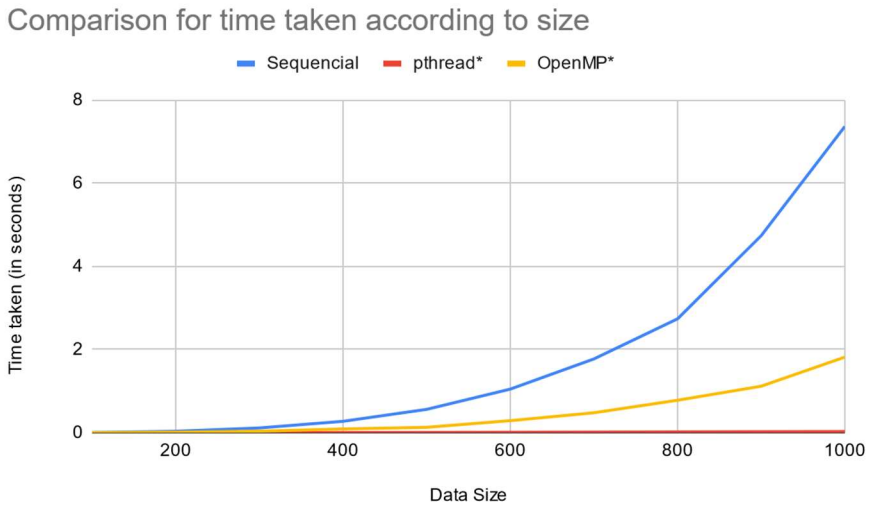
Codes are available on attached files.

Comparing methods according to data size with time taken (in seconds):

1. Between 100 and 1000 (with increment of 100)

	Data Size									
	100	200	300	400	500	600	700	800	900	1000
Sequencial	0.003732	0.034201	0.113765	0.273223	0.560687	1.047661	1.773478	2.738366	4.738423	7.364651
pthread*	0.001476	0.00233	0.004036	0.006344	0.008414	0.011101	0.014903	0.021715	0.026158	0.032545
OpenMP*	0.001855	0.011482	0.035044	0.090319	0.129169	0.289267	0.478883	0.780043	1.117878	1.817399

Graph:

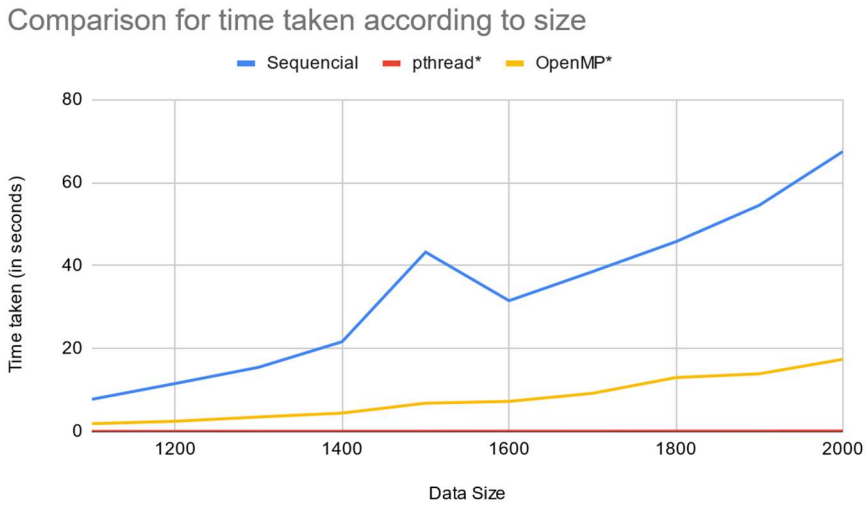


(Note that number of threads for pthread and OpenMP is 3)

2. Between 1000 and 2000 (with increment 100)

	Data Size									
	1100	1200	1300	1400	1500	1600	1700	1800	1900	2000
Sequencial	7.75771	11.5684	15.48368	21.64672	43.24824	31.52976	38.53697	45.75781	54.54315	67.48608
pthread*	0.03323	0.03773	0.04671	0.0529	0.06195	0.07866	0.08015	0.0917	0.10155	0.11529
OpenMP*	1.88932	2.46582	3.49979	4.44066	6.82768	7.25874	9.20699	13.00315	13.91044	17.40428

Graph:

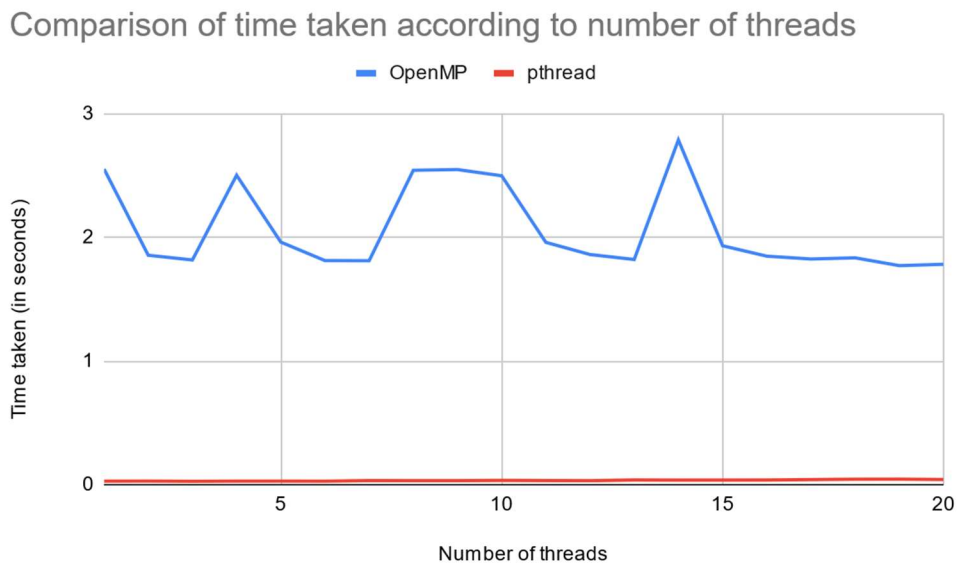


Comparing two parallel programming according to number of threads with time taken (in seconds):

	Threads									
	1	2	3	4	5	6	7	8	9	10
OpenMP	2.552725	1.855395	1.817399	2.502756	1.96197	1.812312	1.811139	2.542697	2.548511	2.498216
pthread	0.028334	0.02898	0.027117	0.028567	0.028782	0.027998	0.033052	0.032301	0.03238	0.034637

	Threads									
	11	12	13	14	15	16	17	18	19	20
OpenMP	1.959771	1.861374	1.821142	2.787854	1.932357	1.848093	1.825128	1.834643	1.77146	1.782356
pthread	0.033216	0.032083	0.037863	0.036779	0.03714	0.037486	0.04103	0.044165	0.044618	0.041603

Graph:



Based on my findings:

1. Using parallel programming does improve performance on matrix multiplication, which is how it must be compared with sequential programming, more than half of the execution time. However, pthread improves the performance significantly compared with OpenMP implementation. The pthread and OpenMP worked perfectly fine. I tested it, I print the first 5x5 elements after the multiplication result, and both worked perfectly fine.
    a. Proof of pthread works:

```
$ ./matrixMultiply_pthread.exe
8096 8510 9206 10578 11670
1380 2691 3111 4287 9435
4140 7728 7920 16936 19744
5520 11316 12432 22134 23746
3772 6049 6745 13507 16783
Time taken by function: 37800 microseconds
```

    b. Proof of OpenMP works:

```
$ ./matrixMultiply_openmp.exe
2550 3040 4010 4588 4906
3650 5400 6340 7530 8052
250 495 535 1980 2016
1050 1470 2280 2586 2622
900 4260 4500 5197 5689
Time taken by function: 2113189 microseconds
```

2. Size of the matrices also affects the performance. In both size 100 x 100 and 1000 x 1000, pthread has the lowest execution time. In addition, we also learn that the time taken rises exponentially. However, we can only see that in sequential and OpenMP methods, pthread grows, but on a much smaller scale.
3. Number of threads also affects the performance for parallel programming. For pthread, it shows that at some point, increasing number of threads will increase its execution time.

It goes as well for OpenMP, but it’s more unpredictable. At some point, it will have lower execution time when you increase number of threads. Based on my findings, the unpredictability of OpenMP caused by sharing of variables. OpenMP also works perfectly fine.