Student Name: Rakyan Adhikara

Student ID: 219548135

Task M2.T1P (Parallel Matrix Multiplication)

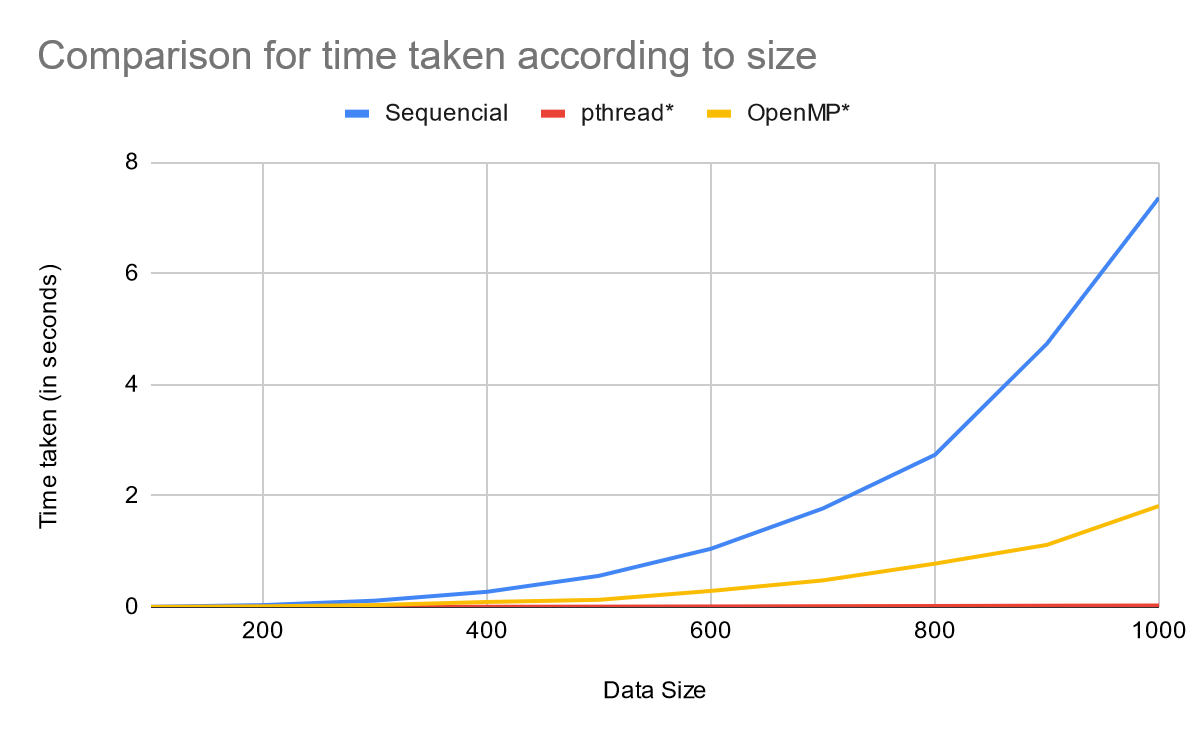
Codes are available on attached files.

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

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

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 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\* | 0.002065 | 0.003276 | 0.003193 | 0.00422 | 0.008572 | 0.012157 | 0.020535 | 0.019342 | 0.029985 | 0.030427 |
| OpenMP\* | 0.001855 | 0.011482 | 0.035044 | 0.090319 | 0.129169 | 0.289267 | 0.478883 | 0.780043 | 1.117878 | 1.817399 |

Graph:

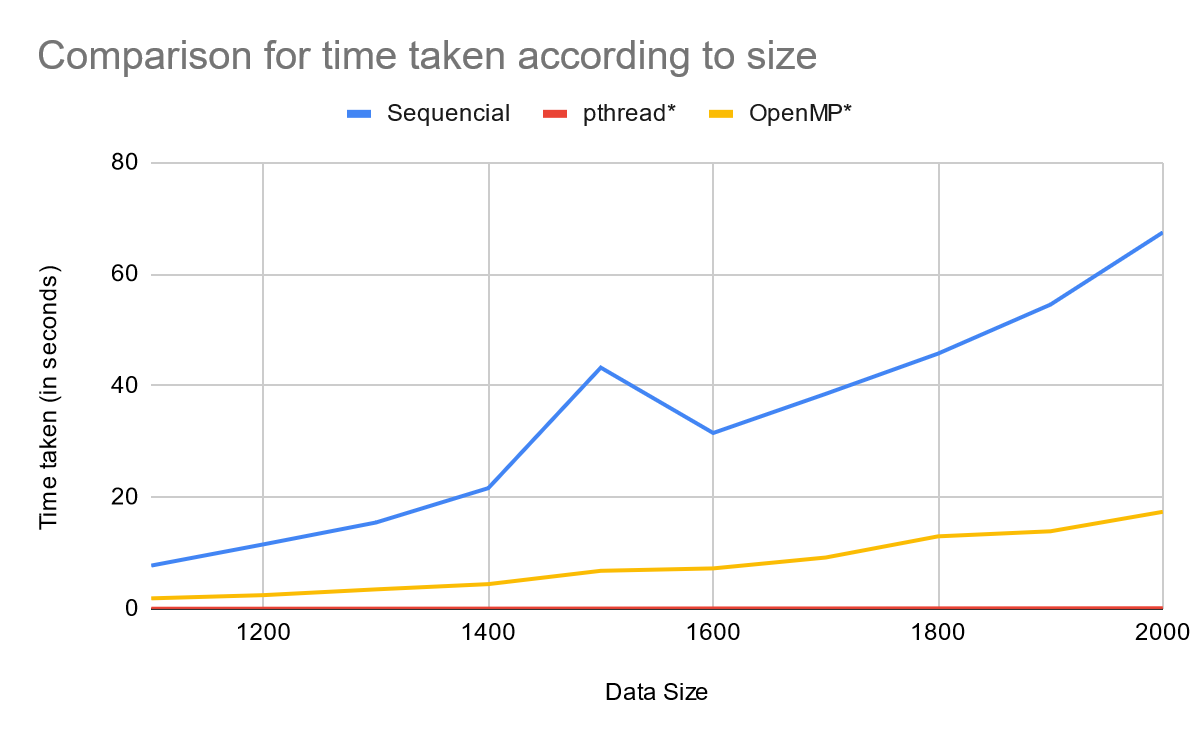


(Note that number of threads for pthread and OpenMP is 3)

1. Between 1000 and 2000 (with increment 100)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1100 | 1200 | 1300 | 1400 | 1500 | 1600 | 1700 | 1800 | 1900 | 2000 |
| Sequencial | 7.75771 | 11.568381 | 15.48368 | 21.646721 | 43.248238 | 31.529757 | 38.53697 | 45.757814 | 54.543146 | 67.486083 |
| pthread\* | 0.017937 | 0.021276 | 0.046706 | 0.026614 | 0.042508 | 0.049003 | 0.05744 | 0.065851 | 0.072808 | 0.082814 |
| OpenMP\* | 1.889317 | 2.465816 | 3.499787 | 4.440655 | 6.827678 | 7.258742 | 9.206989 | 13.003153 | 13.910438 | 17.404277 |

Graph:

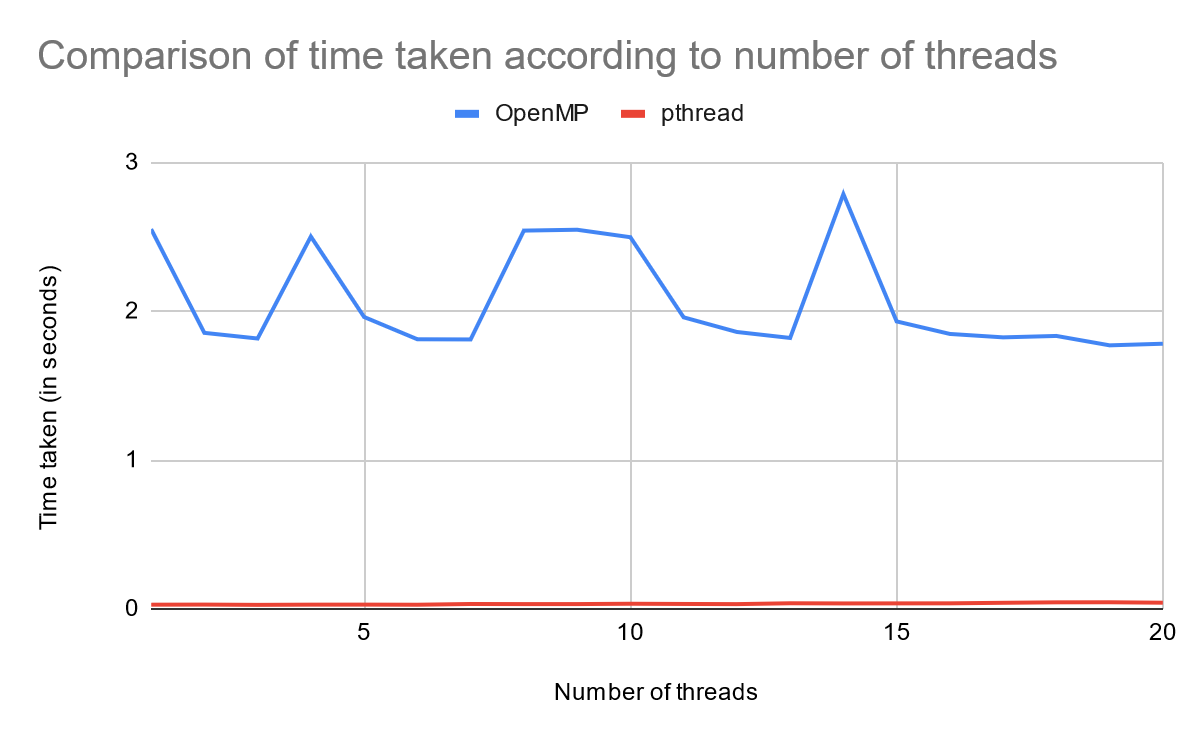


Comparing two parallel programming according to number of threads with time taken (in seconds):

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 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 | 0.009433 | 0.01299 | 0.017659 | 0.019431 | 0.018154 | 0.034981 | 0.043929 | 0.049737 | 0.052952 | 0.062121 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Threads | | | | | | | | | |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| OpenMP | 1.959771 | 1.959771 | 1.959771 | 1.959771 | 1.959771 | 1.959771 | 1.959771 | 1.959771 | 1.959771 | 1.959771 |
| pthread | 0.063263 | 0.069237 | 0.076856 | 0.083469 | 0.086193 | 0.094364 | 0.100766 | 0.105706 | 0.109815 | 0.125877 |

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.
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.