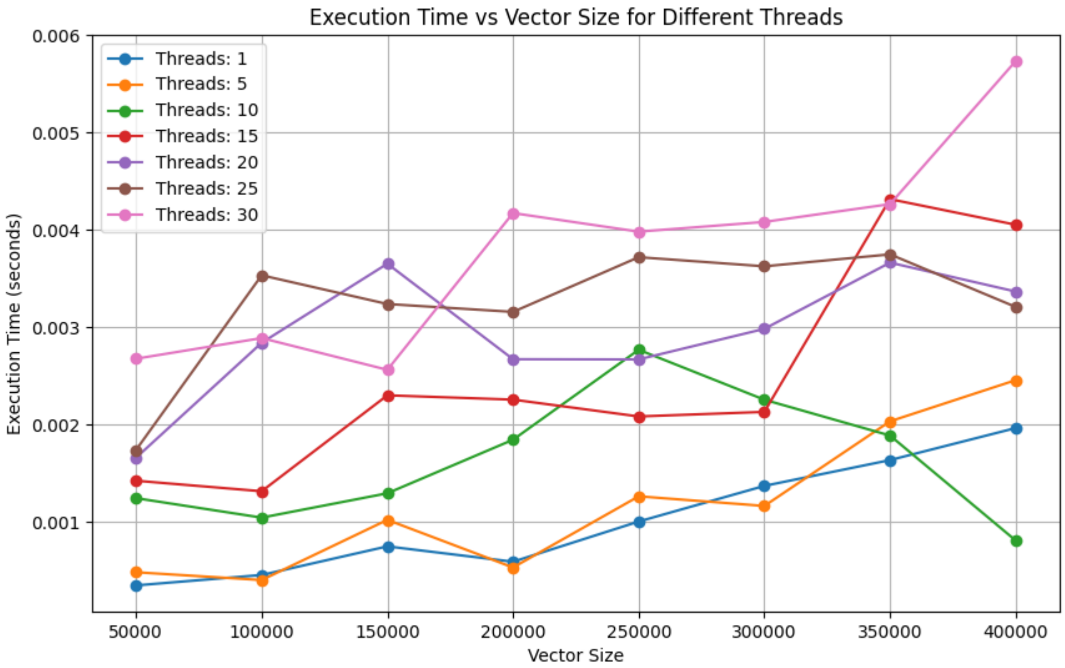
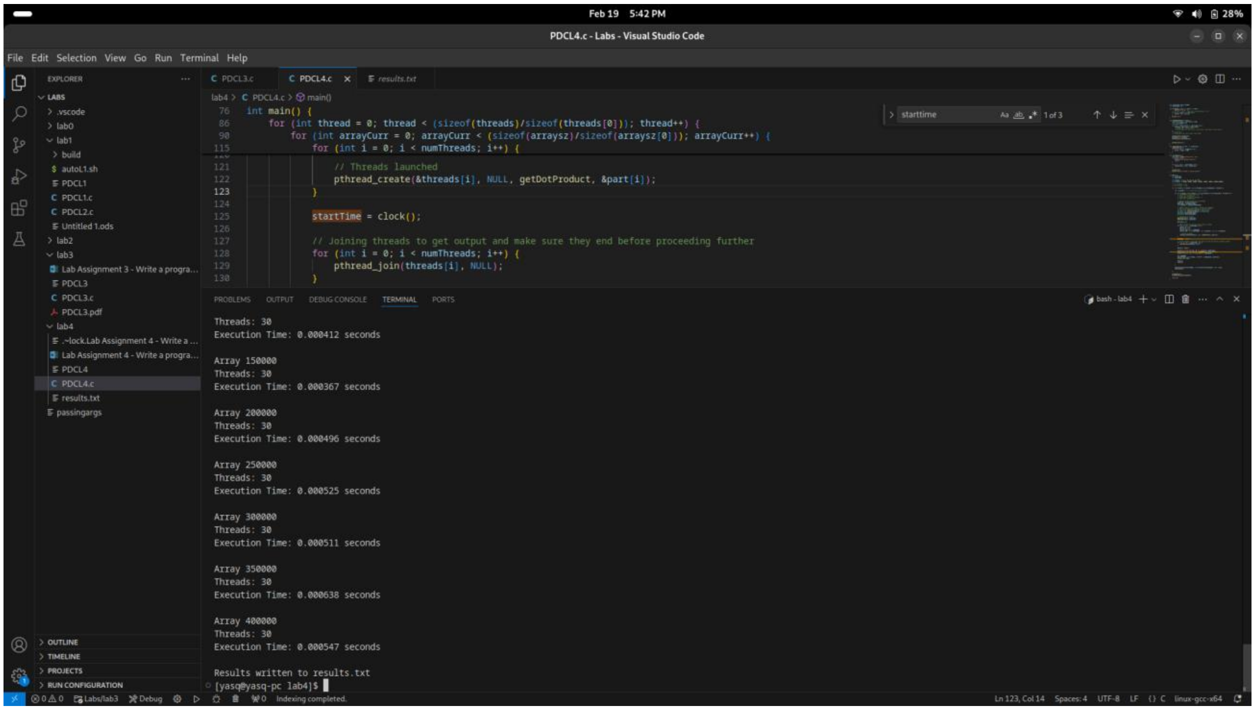
Parallel and Distributed Computing
Lab 4
Shazain
27115

Graph and table for execution time.
Time recorded before creating threads



Threads ↓ / Array Size →	50,000	100,000	150,000	200,000	250,000	300,000	350,000	400,000
1	0.000351	0.000458	0.000752	0.000594	0.001007	0.001373	0.001638	0.001967
5	0.000486	0.000406	0.001024	0.000535	0.001266	0.001167	0.002036	0.002459
10	0.001247	0.001047	0.001297	0.001847	0.002771	0.002258	0.001890	0.000812
15	0.001426	0.001318	0.002303	0.002259	0.002086	0.002133	0.004318	0.004055
20	0.001662	0.002843	0.003655	0.002674	0.002672	0.002986	0.003666	0.003369

25	0.001742	0.003536	0.003241	0.003160	0.003722	0.003628	0.003751	0.003211
30	0.002679	0.002891	0.002565	0.004175	0.003984	0.004084	0.004267	0.005736


Feb 19 5:42 PM
PDCL4.c - Labs - Visual Studio Code
File Edit Selection View Go Run Terminal Help
PDCL3.c
PDCL4.c
results.txt
lab4 > PDCL4.c > main()
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Indexing completed.

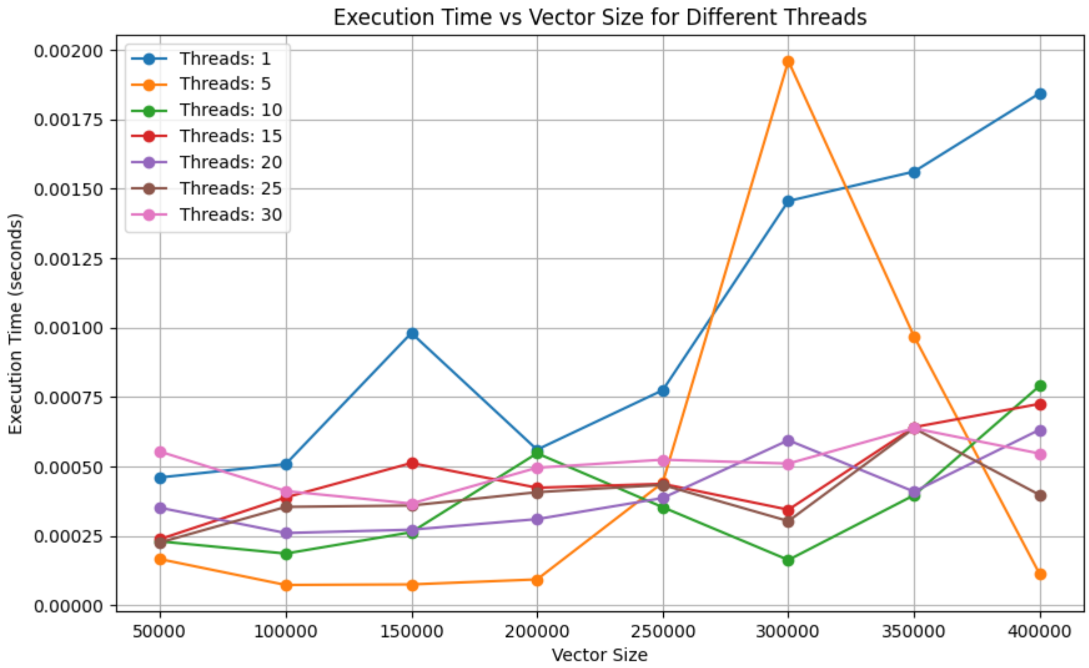
```c
int main() {
    for (int thread = 0; thread < (sizeof(threads)/sizeof(threads[0])); thread++) {
        for (int arrayCurr = 0; arrayCurr < (sizeof(arraysz)/sizeof(arraysz[0])); arrayCurr++) {
            for (int i = 0; i < numThreads; i++) {
                // Threads launched
                pthread_create(&threads[i], NULL, getDotProduct, &part[i]);
            }

            startTime = clock();

            // Joining threads to get output and make sure they end before proceeding further
            for (int i = 0; i < numThreads; i++) {
                pthread_join(threads[i], NULL);
            }
```

```
Threads: 30
Execution Time: 0.000412 seconds

Array 150000
Threads: 30
Execution Time: 0.000367 seconds

Array 200000
Threads: 30
Execution Time: 0.000496 seconds

Array 250000
Threads: 30
Execution Time: 0.000525 seconds

Array 300000
Threads: 30
Execution Time: 0.000511 seconds

Array 350000
Threads: 30
Execution Time: 0.000638 seconds

Array 400000
Threads: 30
Execution Time: 0.000547 seconds

Results written to results.txt
[yasq@yasq-pc lab4]$
```

The vector dot product computation is parallelized by distributing the workload between multiple threads. The total vector size is divided equally among the available threads, according to the number of threads and the vector size. Each thread is assigned a start and end index, ensuring that they work on their section of the vectors only. This approach allows multiple threads to perform computations simultaneously, reducing execution time compared to a single-threaded approach.

Each thread computes the dot product for its assigned portion by iterating through its chunk and calculating the sum of element-wise multiplications. A mutex is used to ensure safe updates to the global dot product variable, preventing race conditions when multiple threads try to modify it at the same time. After all threads finish execution, the results are accumulated in the global variable, and the final dot product value is printed. Thread synchronization using pthread_join() ensures that all computations are completed before displaying the result. This method efficiently leverages multithreading to improve performance while maintaining accurate results.

Time recorded before joining threads



Threads ↓ / Array Size →	50,000	100,000	150,000	200,000	250,000	300,000	350,000	400,000
1	0.000461	0.000509	0.000981	0.000561	0.000775	0.001456	0.001562	0.001844
5	0.000167	0.000074	0.000076	0.000094	0.000442	0.001960	0.000969	0.000113
10	0.000231	0.000187	0.000264	0.000548	0.000354	0.000164	0.000397	0.000791
15	0.000239	0.000389	0.000513	0.000424	0.000438	0.000345	0.000642	0.000726
20	0.000352	0.000261	0.000273	0.000311	0.000387	0.000595	0.000410	0.000633
25	0.000227	0.000355	0.000360	0.000408	0.000434	0.000304	0.000638	0.000398
30	0.000554	0.000412	0.000367	0.000496	0.000525	0.000511	0.000638	0.000547