Parallel Distributed Computing
Assignment 1
Report

I210844
Abdullah Chaudhary
CS-B

Findings:

After doing this assignment, I came to the conclusion that to save the processing time; It is very essential to use the threads. As the sequential processes only work in one thread, they rely on one core for all the processing and takes too much computational time.

It was not very evident in this assignment because the dataset was very small and the over-head to divide the computation to each core has its own computational time. Because of this, the total time consumed doesn't show such a big difference. But once the dataset is large, The total time consumed by the program run by different threads is exceptionally small.

- ### Sequential Process

  The sequential process was working in one thread and waiting for a loop or function to execute to process further with the program hence taking a large amount of executional time.

- ### Multi-threaded Execution (OS-Assigned Affinity)

  After doing the second part of the assignment, I understood how the threads work and how one process can divide its work into different threads and they run concurrently so that the program doesn't have to wait for computation or loop to execute to process further with the program. It reduces the computational time greatly. And it is easier to implement as the OS takes the responsibility to assign the tasks to the threads but it gives us lesser control over the program execution.

- Multi-threaded Execution (User-Assigned Affinity)

Finally, After implementing the multiple threads by user assigned affinity, I understand how we can assign the threads to the core ourselves which gives us greater control over the usage of the cores. If you want to manually assign the threads to the cores, you have to use this approach. It is a bit harder to implement but we have the control and we can improve time consumption drastically by ourselves.

Approach

- Sequential Process
    1. For the sequential process, what I basically did is that I first read the input array and the number of iterations from the file.
    2. Then I transformed the array according to the formula given to the intermediate array.
    3. Then I computed all the permutations of the input array that provided me with a 2D permutation matrix. I took the dot product of the permutation matrix with the input array to generate a weight array that was the size of the factorial of the size of the input array.
    4. After getting the weights array, I applied a loop that runs till the size of the iterations we read from the file. We computed the output array by taking the sum of the dot product of the weights with the intermediate array.
    5. In the final step, we assign the output array to the input array and then generate a permutation matrix according to it. Then we generate weights from it and compute the weight loss and the iterations go on and on.
    6. Finally, we free up extra space taken by the arrays and print out the final output array.

- Multi-threaded Execution (OS-Assigned Affinity)

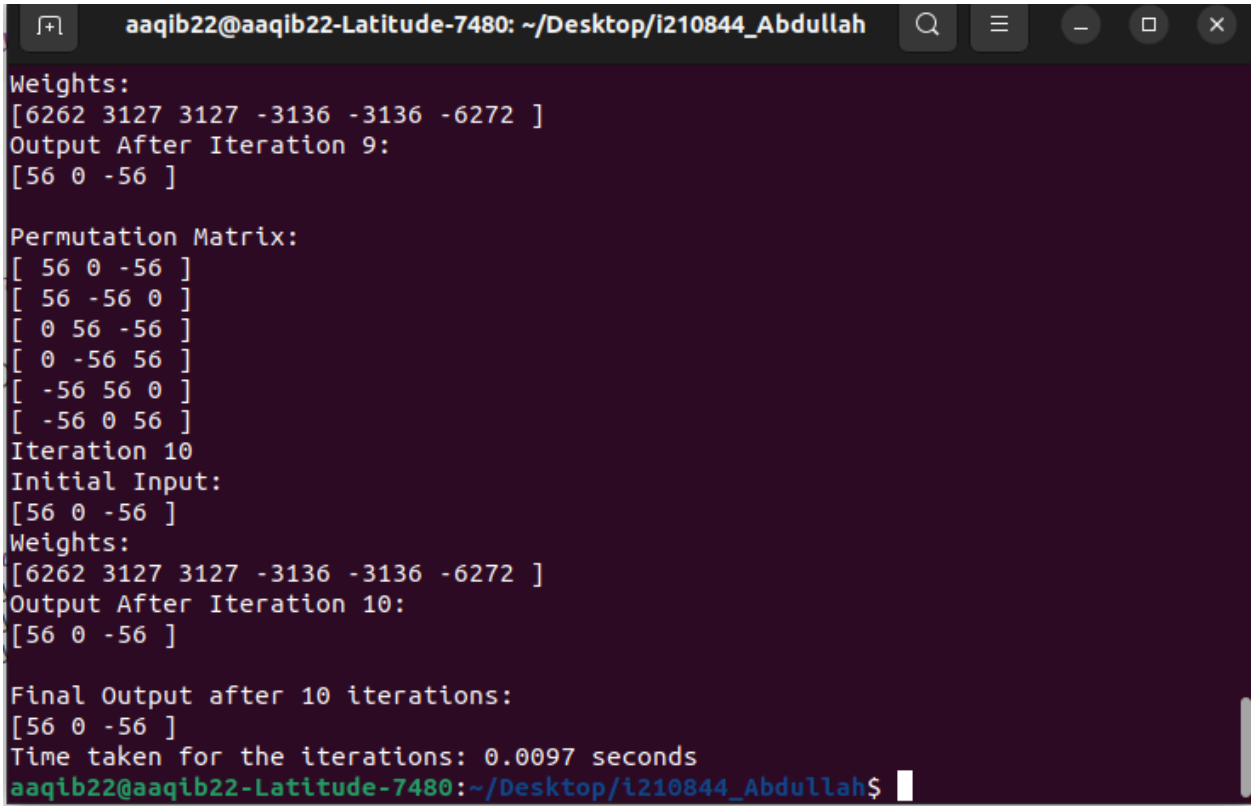
The approach is the same in this step but I created a structure to pass to the thread function which includes the arrays involved in the function that was computed by the threads.I parallelized the process by implementing multiple threads which were assigned to the cores by the OS itself.

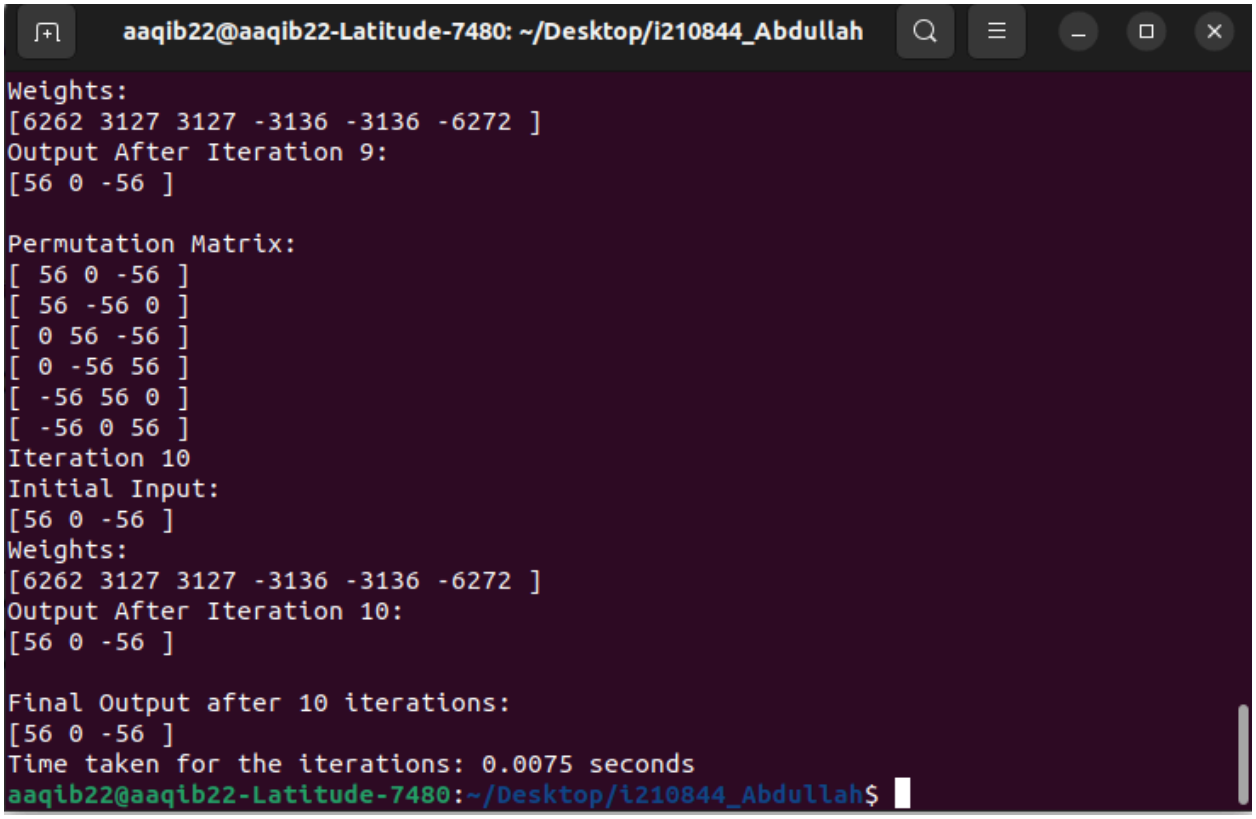
- Multi-threaded Execution (User-Assigned Affinity)

    The approach is the same in this step but the program will ask the user about which core he wants to assign the thread too. The program will ask the user as to which core he wants to assign the thread too. Then the program will assign that thread to that specific core which will give us greater control over the core assignment.

Timing Comparison

As we can see in these images, there is a difference in the time between each process. The difference is not much drastic because the data set is very small and when threads are created there is an overhead computational time for it too. As we increase the dataset, the time difference will drastically increase too.


aaqib22@aaqib22-Latitude-7480: ~/Desktop/i210844_Abdullah
Weights:
[6262 3127 3127 -3136 -3136 -6272]
Output After Iteration 9:
[56 0 -56]
Permutation Matrix:
[56 0 -56]
[56 -56 0]
[0 56 -56]
[0 -56 56]
[-56 56 0]
[-56 0 56]
Iteration 10
Initial Input:
[56 0 -56]
Weights:
[6262 3127 3127 -3136 -3136 -6272]
Output After Iteration 10:
[56 0 -56]
Final Output after 10 iterations:
[56 0 -56]
Time taken for the iterations: 0.0097 seconds
aaqib22@aaqib22-Latitude-7480:~/Desktop/i210844_Abdullah$


aaqib22@aaqib22-Latitude-7480: ~/Desktop/i210844_Abdullah
Weights:
[6262 3127 3127 -3136 -3136 -6272]
Output After Iteration 9:
[56 0 -56]
Permutation Matrix:
[56 0 -56]
[56 -56 0]
[0 56 -56]
[0 -56 56]
[-56 56 0]
[-56 0 56]
Iteration 10
Initial Input:
[56 0 -56]
Weights:
[6262 3127 3127 -3136 -3136 -6272]
Output After Iteration 10:
[56 0 -56]
Final Output after 10 iterations:
[56 0 -56]
Time taken for the iterations: 0.0075 seconds
aaqib22@aaqib22-Latitude-7480:~/Desktop/i210844_Abdullah$

```
Weights:
[6262 3127 3127 -3136 -3136 -6272 ]
Output After Iteration 9:
[56 0 -56 ]

Permutation Matrix:
[ 56 0 -56 ]
[ 56 -56 0 ]
[ 0 56 -56 ]
[ 0 -56 56 ]
[ -56 56 0 ]
[ -56 0 56 ]
Iteration 10
Initial Input:
[56 0 -56 ]
Weights:
[6262 3127 3127 -3136 -3136 -6272 ]
Output After Iteration 10:
[56 0 -56 ]

Final Output after 10 iterations:
[56 0 -56 ]
Time taken for the iterations: 0.0020 seconds
aaqib22@aaqib22-Latitude-7480:~/Desktop/i210844_Abdullah$
```