**Assignment 2**

**LU Decomposition**

**Using Pthread and OpenMP**

**By**

**Ashutosh Mohanta**

**2019PH10620**

**Working:**

**Serial:**

To compile and run serial.c:

gcc serial.c -lm

time ./a.out n thread\_count

Where n\*n is the size of matrix. For serial code the thread\_count won’t matter as it will only use 1 core. The program takes it as the argument just for the sake of uniformity.

**example:**

gcc serial.c -lm

time ./a.out 7000 1

**Pthread:**

To compile and run pthread.c:

gcc pthread.c -lpthread -lm

time ./a.out n thread\_count

Where n\*n is the size of matrix. thread\_count can be atmost 32.

**OpenMP:**

To compile and run pthread.c:

gcc openmp.c -fopenmp -lm

export OMP\_NUM\_THREADS=thread\_count

time ./a.out n thread\_count

example:

export OMP\_NUM\_THREADS=8

time ./a.out 7000 8

Where n\*n is the size of matrix. thread\_count can be anything.

**Graphs:**

As we can see the minimum time takes is around 4 threads as I ran the code on 4 core laptop. After the 4 cores the speed tend to stay constant and/or decrease a bit.

**SpeedUp (2 Cores) = 1.94**

**SpeedUp (4 Cores) = 3.40**

**SpeedUp (8 Cores) = 3.33**

**Algorithm Details:**

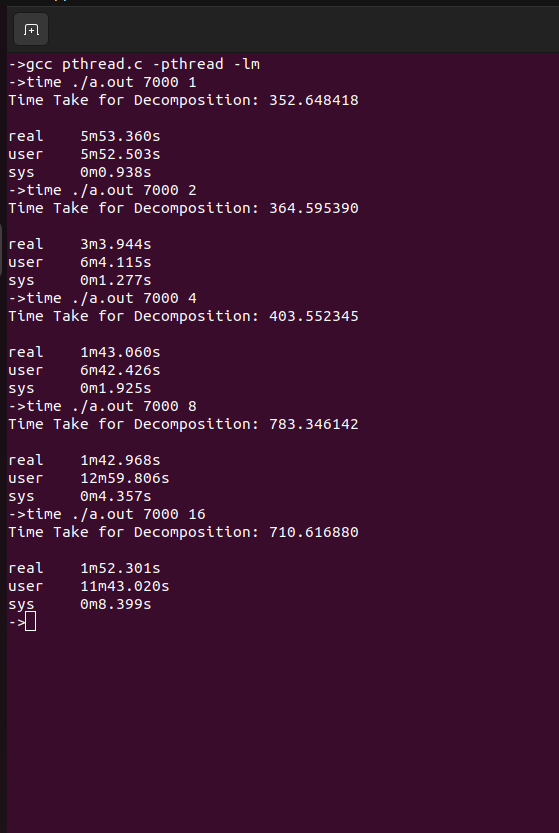
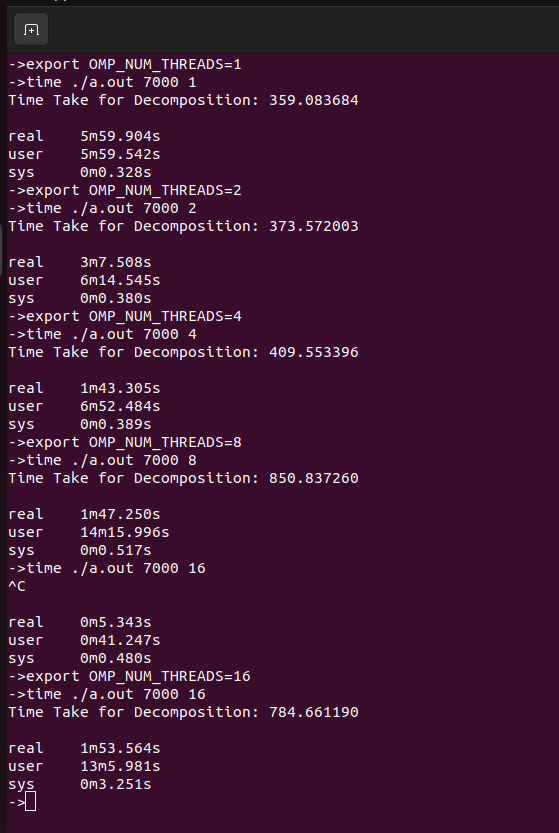
**Pthread:**

I used row division method to divide chunks of rows among threads. For Synchronization I used pthread\_join to wait for synchronization. I also used mutex wherever necessary to ensure there is no race condition.

**OpenMP:**

Using #pragma to guide the compiler to divide the for loops among the threads. Four of the for loops are parallelized in this manner.

**Timing Details:**

****

I considered “real time” instead of “Time Take for Decomposition”, as the latter sums the times of all the cores and doesn’t reflect the real time that the user has to wait.