# Operating System Project Report

# “pthread vs OpenMP”

# *Section – B*

# Usman Khan (16k-3733)

# Taha Ahmed(16k-3777)

# Aqrab Raza (16k-3685)

**Objective:**

The objective in making this project was set to attain the time complexity differences while performing two simple sorting algorithms over a randomly generated array of variable sizes, using the pthread.h and omp.h library on C language. Furthermore, the project keeps both the times calculated of each library and compares them together over a chart.

**Introduction:**

* **Threads**:

Threads are part of a process and can be called a light weight process. Threads are used to perform tasks which are independently different and they share address space and global variables and are independent.

* **OpenMP:**

A multithread programming can be driven by using the application programming interface called OpenMP, which works its best at the memory sharing parallelism using its own compiler directives, environment variables and the runtime library routines.

**Programming Platform used:**

The programs are scripted in C language using ‘gedit’ on Linux Ubuntu 16.04 LTS.

**Methodology:**

The project is based upon two .c extension files, namely pthread.c and omp.c. The file **pthread.c** makes two different threads, **shellThread** and **bubbleThread.** These two threads call the functions **shellSort( )** and **bubbleSort( )** respectively. Both the functions generate random numbers *(which are to be edited into the script as per time calculation consideration)* sorts them in accordance to the respective sorting algorithm types. Inside main(), the time variables then calculate the time differences and print the time taken by each algorithm along with the number of elements it was compiled with.

On the other hand, the file omp.c is scripted inside a single main function. The function first generates random variables as per the given size inside for loop to be fed inside a dynamic array. In line #29 the starting time for shell sort is set for sorting in ascending order using **OpenMP**, the sorting algorithm then runs from line #32 till line #57 and then the ending time is noted. The difference is calculated and displayed along with the number of elements given. Further in the code of **OpenMP** method, the bubble sort algorithms run all over the same way and after sorting, it prints the total sorting time taken by bubble sort to sort the same data in descending order.

**Application:**

All the applications using multiprogramming and multithreading over multicore processors where the **OpenMP & pthread** libraries handle the program dividing it among threads for example, creating multiple applications over a GPU or modern multithreading, implementing, testing, anddebugging multithreaded Java and C++/Pthreads/Win32 programs. It can also be used to generated series or sequences. Computer clusters run parallel applications at real-time made over openMP.

**pthread – OpenMP time complexity comparison table**

We ran the pthread and openMP programs on various number of arrays to compute the running time of each algorithm in milli-seconds.

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithm time (ms) | Number of array elements | | |
|  | 5000 | 20000 | 100000 |
| pthread\_ShellSort | 0.008 | 0.021 | 0.035 |
| pthread\_BubbleSort | 0.152 | 1.585 | 34.23 |
| pthread\_QuickSort | 0.006 | 0.007 | 0.015 |
| OpenMP\_ShellSort | 0.001 | 0.01 | 0.052 |
| OpenMP\_BubbleSort | 0.057 | 1.248 | 29.71 |
| OpenMP\_QuickSort | 0.001 | 0.003 | 0.01 |