CSE557: PROJECT REPORT
TOPIC: MEAN SHIFT ALGORITHM USING OPENMP AND CUDA

INTRODUCTION:

Mean Shift algorithm is an unsupervised clustering algorithm that aims to discover blobs in a smooth density of samples. It is a centroid-based algorithm that works by updating candidates for centroids to be the mean of the points within a given region.The higher density regions correspond to regions with more data-points and lower density regions correspond to regions with fewer points. The algorithm is compared with K-Means clustering algorithm and also has many use cases in the field of Machine Learning and Computer Vision related projects. The current project implements Mean Shift algorithm using various techniques such as sequential, OpenMp and CUDA to compare the time taken by each of the methods to find the centroids from a given number of points. The implementations have been executed with points with dimension 2d and 3d.

IMPLEMENTATION:

The algorithm has been implemented in three techniques which are Sequential, OpenMp and CUDA. The algorithm has been executed on a series of datasets with different number of points in each set and with the points being in both 2D and 3D dimensions.The algorithm successfully gets the centroids in each of the dataset and also ensures that the results in each of the implementations remains consistent.The algorithm used for sequential implementation and also on which the OpenMp and CUDA implementations are based is given below.

Algorithm:

```
Input: $\mathcal{S} \subset \mathbb{R}^D$. $I \in \mathbb{N}$. $\sigma, R, \delta \in \mathbb{R}$.
Input: Optional: $\epsilon \in \mathbb{R}$.
Output: Centroids $\{\boldsymbol{\mu}_1, \dots \boldsymbol{\mu}_k\}$ for some $k \in \mathbb{N}$
for $t = 1, \dots, I$ do
    for $i = 1, \dots, N$ do
        $\boldsymbol{\tau} = \mathbf{0}$
        $\eta = 0$
        for $j = 1, \dots, N$ do
            if $\|\mathbf{x}_i^{(t)} - \mathbf{x}_j^{(t)}\|^2 \le R$ then
                $\boldsymbol{\tau} = \boldsymbol{\tau} + K_\sigma(\mathbf{x}_i^{(t)} - \mathbf{x}_j^{(t)})\mathbf{x}_j^{(t)}$
                $\eta = \eta + K_\sigma(\mathbf{x}_i^{(t)} - \mathbf{x}_j^{(t)})$
            end if
        end for
        $\mathbf{x}_i^{(t+1)} = \boldsymbol{\tau}/\eta$
        if $\|\mathbf{x}_i^{(t+1)} - \mathbf{x}_i^{(t)}\| \le \epsilon$ then
            stop_shifting($\mathbf{x}_i^{(t+1)}$)
        end if
    end for
end for
$\{\boldsymbol{\mu}_1, \dots \boldsymbol{\mu}_k\} \leftarrow$ reduce_centroids($\mathcal{S}^{(I)}, \delta$)
return $\{\boldsymbol{\mu}_1, \dots \boldsymbol{\mu}_k\}$
```

The make file for each implementation is present in each folder and the code can be compiled using the command 'make' and then can be executed using the commands:

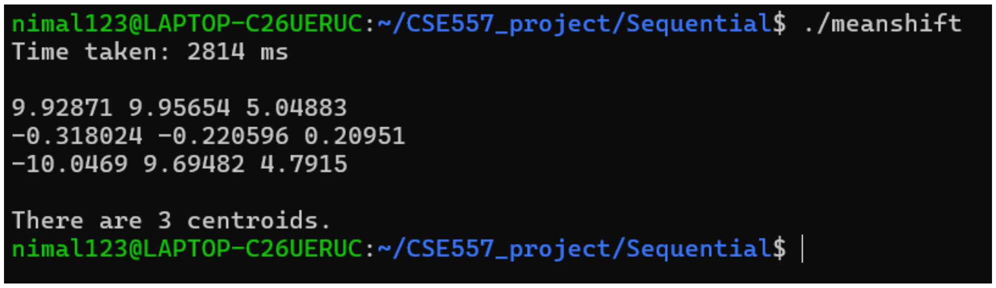
./meanshift ------- For Sequential Implementation

./meanshift_omp -------- For OpenMP Implementation

./cuda ---------- For CUDA Implementation

Sequential:

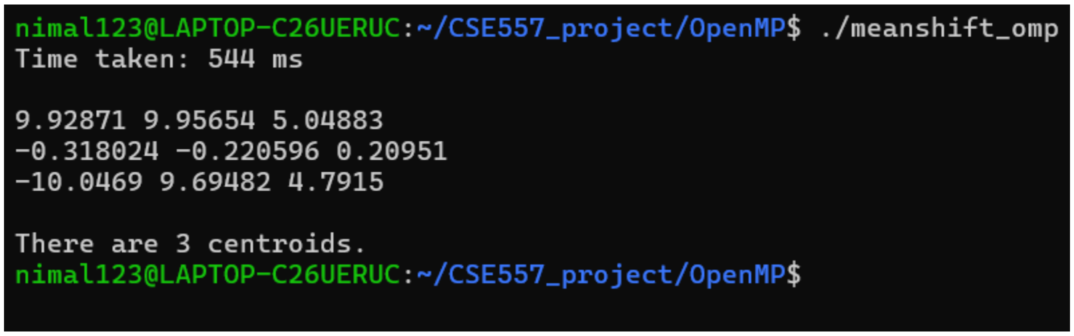
The sequential version is implemented directly from the pseudocode without any optimization techniques. The use of -O3 is the only optimization usd for sequential implementation during compilation. One of the major reasons for implementing sequential version is that it forms a base performance index when comparing the execution time of the results given by the implementations with various optimizations.



```
nimal123@LAPTOP-C26UERUC:~/CSE557_project/Sequential$ ./meanshift
Time taken: 2814 ms

9.92871 9.95654 5.04883
-0.318024 -0.220596 0.20951
-10.0469 9.69482 4.7915

There are 3 centroids.
nimal123@LAPTOP-C26UERUC:~/CSE557_project/Sequential$
```

OpenMp:

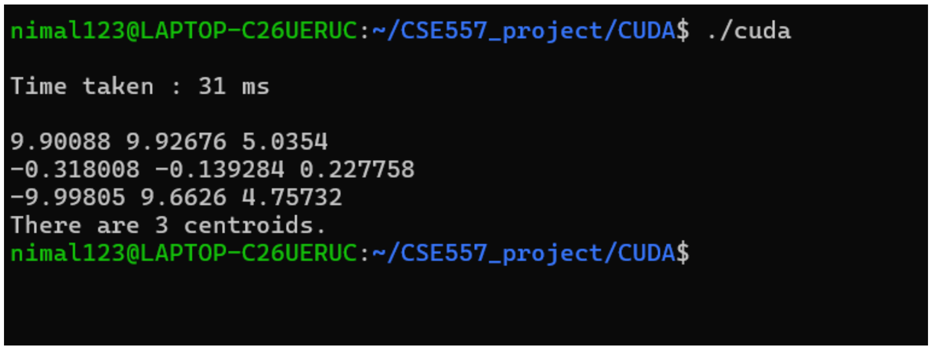
OpenMp is an API which implements multi-threading in a fork-join model.The API consists of a set of compiler directives, library routines and environment variables and this influences the runtime behavior of the implementation.The number of threads is decided at runtime by OpenMp which is dynamic or this can be set by the user which is static. In this project, the OpenMp implementation for MeanShift algorithm is based on the dynamic mechanism for the number of threads. The #pragma omp critical directive is used to avoid a possible race condition as different threads write to the same shared variable.



```
nimal123@LAPTOP-C26UERUC:~/CSE557_project/OpenMP$ ./meanshift_omp
Time taken: 544 ms

9.92871 9.95654 5.04883
-0.318024 -0.220596 0.20951
-10.0469 9.69482 4.7915

There are 3 centroids.
nimal123@LAPTOP-C26UERUC:~/CSE557_project/OpenMP$
```

CUDA:

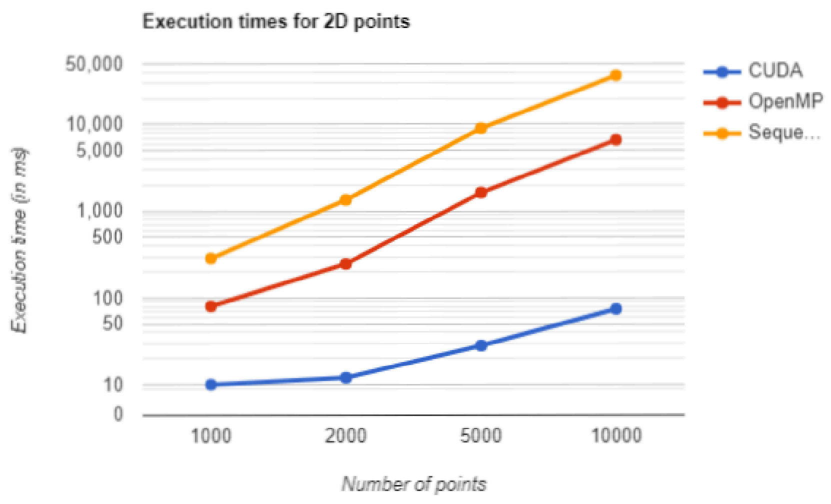
CUDA is a parallel computing platform created by NVIDIA and can be implemented using NVIDIA CUDA-Toolkit. The CUDA platform gives an opportunity for development of general purpose softwares on GPUs. It also gives access to the GPU's instruction set and parallel computational elements. CUDA allows memory accesses as global memory and also shared memory. The implementation in this project is based on global memory access. The CUDA implementation takes advantage of the Single Instruction, Multiple Thread (SIMT) model.

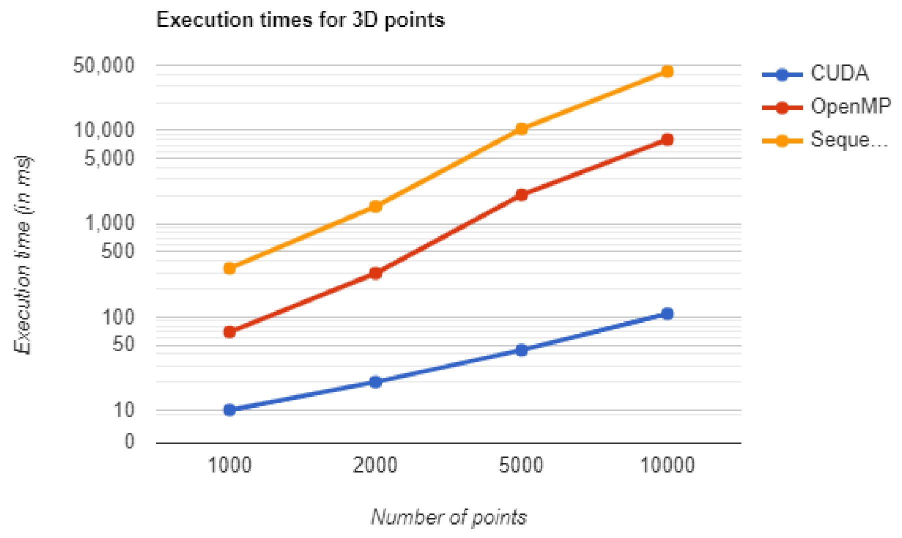


```
nimal123@LAPTOP-C26UERUC:~/CSE557_project/CUDA$ ./cuda

Time taken : 31 ms

9.90088 9.92676 5.0354
-0.318008 -0.139284 0.227758
-9.99805 9.6626 4.75732
There are 3 centroids.
nimal123@LAPTOP-C26UERUC:~/CSE557_project/CUDA$
```

RESULTS:

The implementation has been tested on different datasets with different amounts of points and also with both 2D and 3D dimension points. The CPU used for this analysis is a 4 core Intel i5-9300H. The operating system is Linux/Ubuntu and the GPU used is 4GB Nvidia Geforce GTX 1650. The execution time results have been visually represented with the help of graphs below.





In the above graphs, the execution times of all the three implementations for each dataset is plotted and it is visible that the CUDA implementation with the help of GPU has the least execution time. The OpenMP and Sequential implementation which is based on CPU has much higher execution time and the differences in execution time increases as the number of points in the dataset increases. The difference goes up to 40000 ms between CUDA and sequential implementation in the case of a dataset with 10000 3d points. There is also a slight difference in the centroid coordinates given by each of the implementations even though the difference is negligible. All the three implementations successfully detect the same number of centroids in a given dataset thus showcasing that the accuracy of the algorithm is unhindered even when different optimizations are used. The lower execution time for CUDA implementation also suggests the efficient parallelization of the computation, taking advantage of GPU architecture which is well suited for handling multiple operations simultaneously.

FUTURE WORK:

The algorithm is implemented in three different methods of optimizations but can be further optimized by using shared memory in the CUDA implementation. The shared memory could have slightly better performance compared to the global memory access implementation. The fastest implementation currently which is the CUDA implementation can be tried to implement in various machine learning and computer vision projects and the change in execution time can be compared.

CONCLUSION:

Thus the project focuses on implementation of optimized versions of the mean shift algorithm which is one of the known unsupervised clustering algorithms. Three different methods have been used and the results have been compared and it can be found out that the OpenMp implementation has the fastest execution time and this trend has been tested with a series of datasets with different amounts of points and also different dimensions. Thus the CUDA implementation can be implemented in various other projects involving mean shift algorithm to provide better running time.