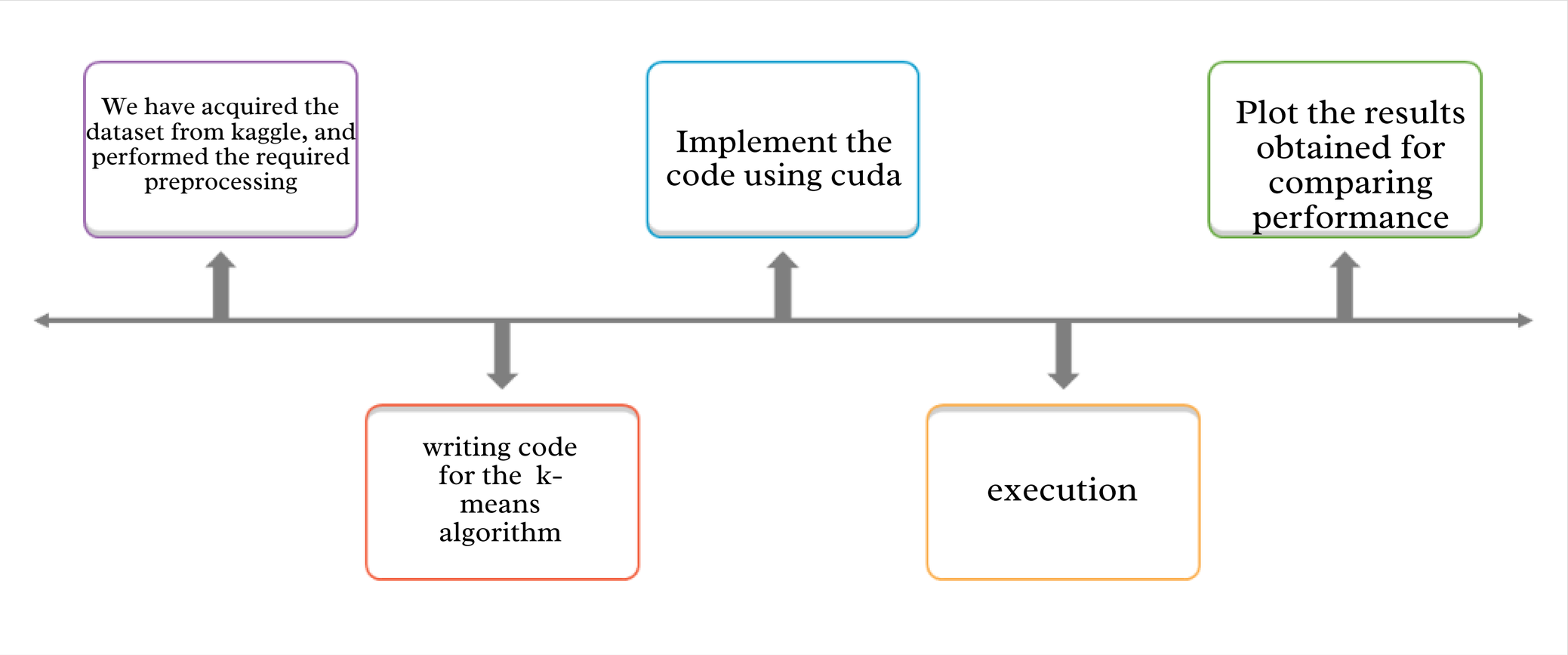
Parallelization of K-means Clustering Model for Brain MRI images using CUDA and OpenMP

- RAMYA SUNDARAM
  21011101097
- SHRUTHI MURUGAN
  21011101124

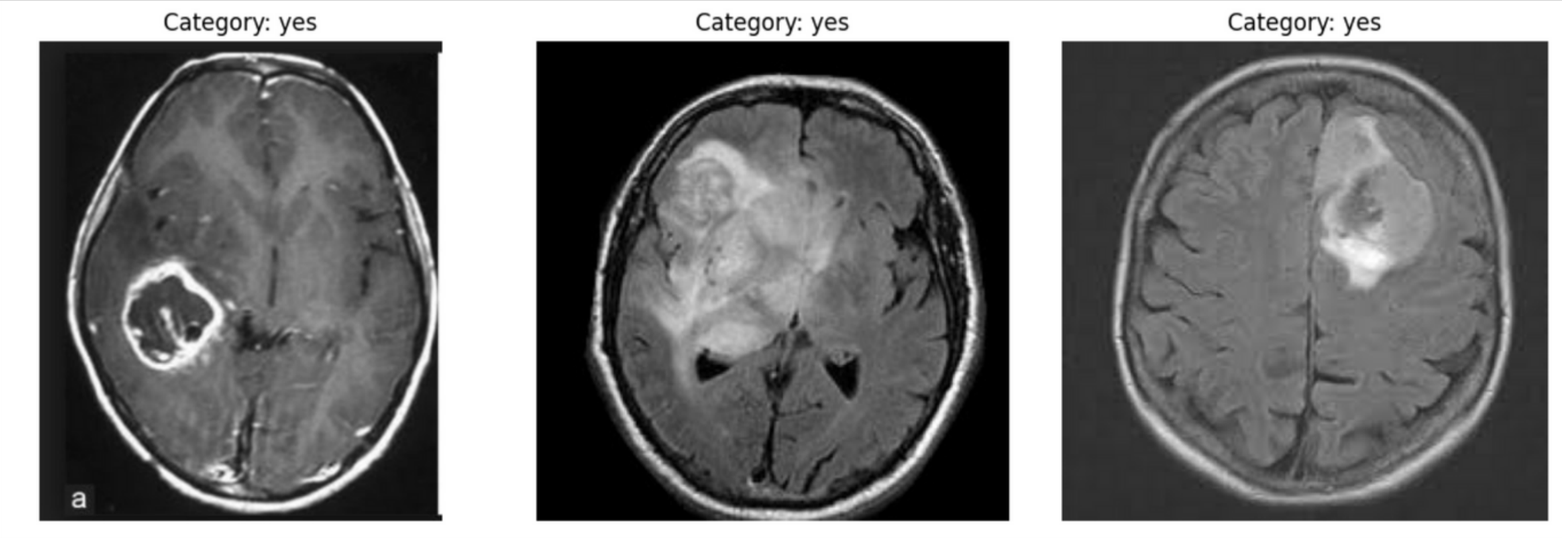
Need for parallelization of K-Means algorithm:

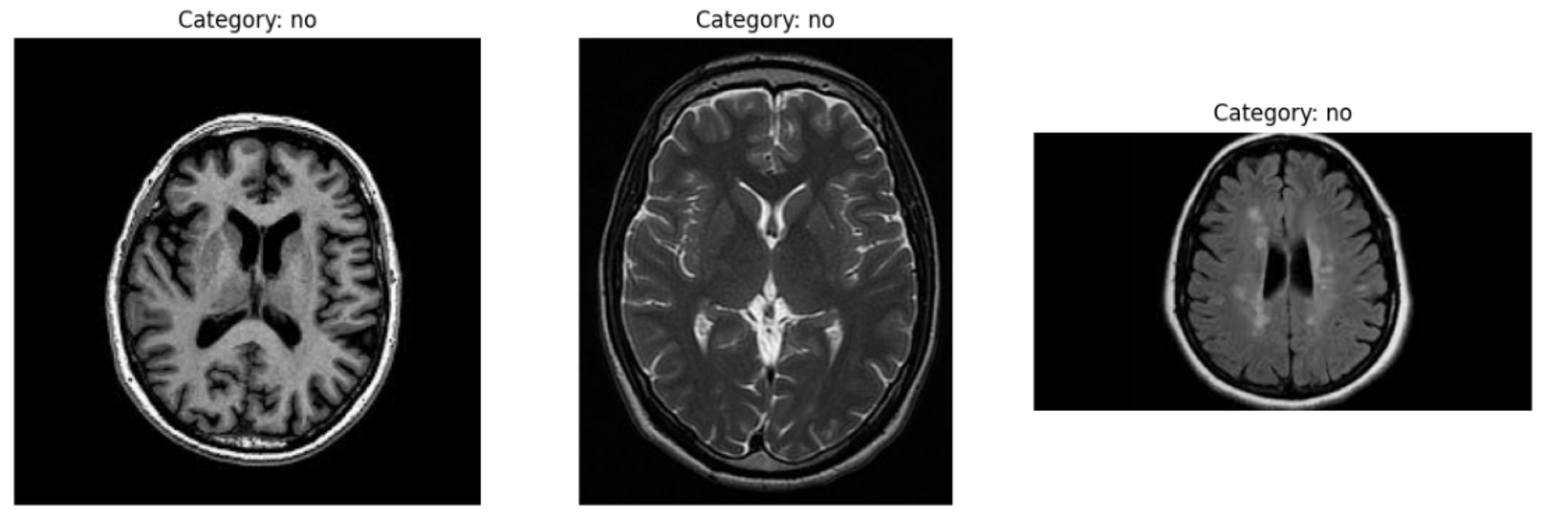
- When dealing with large image datasets, the serial implementation of K-means can be computationally intensive.
- To address this issue and enhance computational efficiency, parallelization techniques utilizing frameworks such as CUDA is employed.
- This parallelization accelerates the computation of centroids, optimizing the overall computational process for the k-means clustering algorithm to identify the clusters of similar features within the MRI dataset.

-TIMELINE TO FINISH:



-Dataset obtained from kaggle: Images with brain tumor and without brain tumor





PREPROCESSING :

- CONVERT TO GRAYSCALE
- RESIZING
- NORMALIZING
- CONVERT IMAGE INTO NUMPY ARRAY
- LOAD PIXEL VALUE OF EACH IMAGE TO 3DDATA.TXT
- USE FOR MODEL

FRAMEWORK USED:

1. **CUDA**

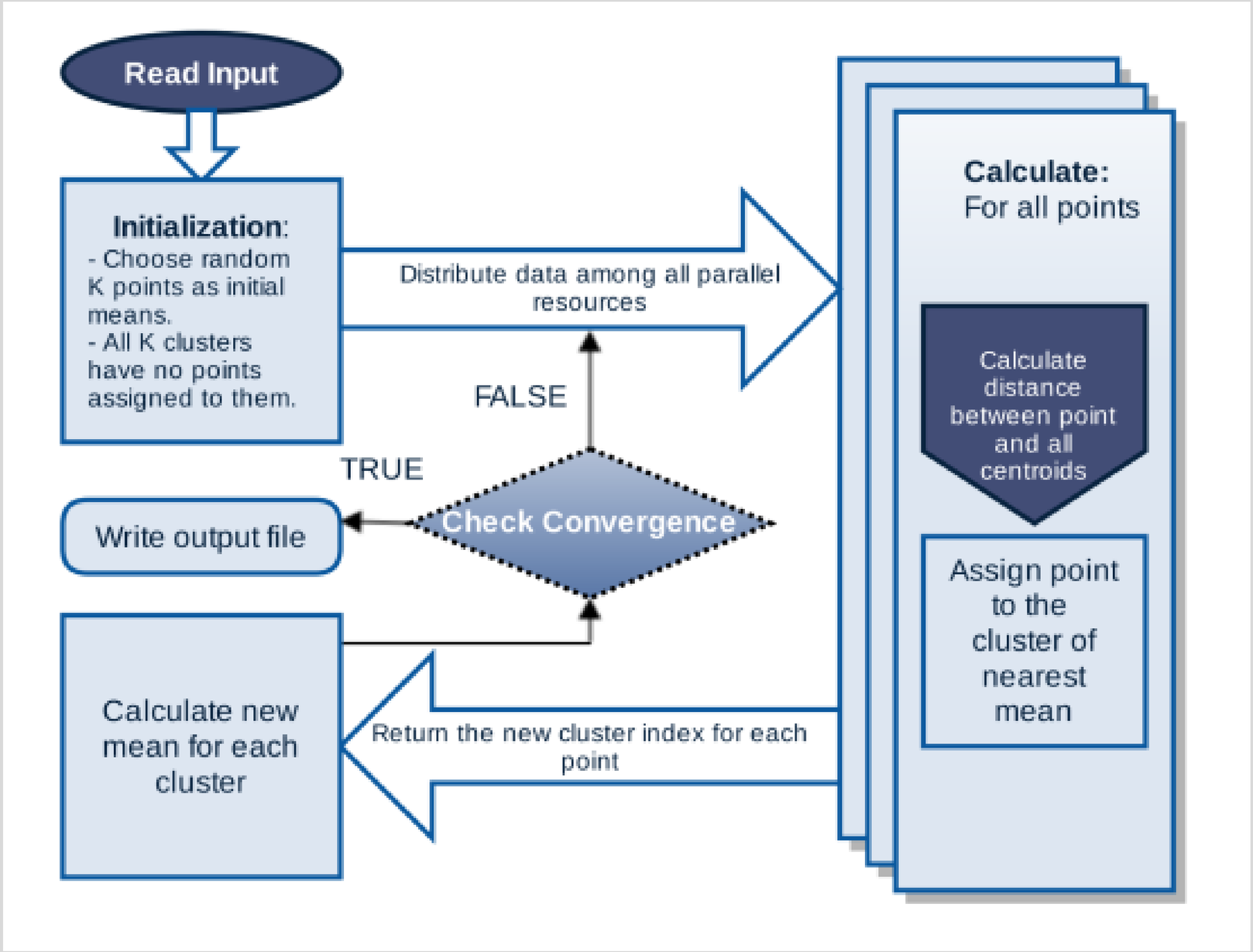
- CUDA is a parallel computing platform and application programming interface (API) that allows software to use certain types of GPU for general purpose processing.

- It is a software layer that gives direct access to the GPU's virtual instruction set and parallel computational elements, for the execution of compute kernels.

2. Open MP :

OpenMP (Open Multi-Processing) is an API (Application Programming Interface) that supports multi-platform shared memory multiprocessing programming in C, C++, and Fortran

Base architecture:



ASPECTS OF PARALLELISM:

1. **Data Parallelism:**

   - Data parallelism involves distributing the workload across multiple processing units(CUDA threads) where each thread operates on a different subset of the input data.
   - In the k-means algorithm, each data point represents a single observation or sample, and the goal is to assign each data point to the nearest cluster centroid.
   - Data parallelism is achieved by assigning each data point to a separate CUDA thread.
   - Within the kmeans_kernel function, the variable i represents the index of the current data point being processed. Each thread computes the distance between its assigned data point and all cluster centroids, and then assigns the data point to the nearest centroid based on this distance.
   - By parallelizing the computation of distances and cluster assignments across multiple threads, the algorithm can process large datasets efficiently.

ASPECTS OF PARALLELISM:

2.Thread Parallelism:

- Thread parallelism refers to the concurrent execution of operations within each CUDA thread, leveraging the massive parallelism available on the GPU.
- Within each CUDA thread, there is a loop that calculates the distance between the data point and each cluster centroid. This loop is responsible for the main computation performed by each thread.
- The distance computation loop is parallelized at the instruction level, meaning that multiple iterations of the loop can be executed simultaneously by different CUDA cores within the same thread.
- This thread-level parallelism allows for efficient computation of distances for each data point, maximizing the utilization of GPU resources.

Approach:

- Take training input images and extracting features of the images in a list as a text file.
- Initialize the CUDA environment. Allocate memory for input data, centroids, clusters, and distances on the GPU (device memory) using CUDA memory management functions.
- Copy the input data from the host (CPU) memory to the device (GPU) memory using cudaMemcpy.
- Define a CUDA kernel function (kmeans_kernel) responsible for calculating distances and assigning points to clusters in parallel.
- Configure grid and block dimensions for parallel execution. Each thread handles a portion of the data.

- Within the CUDA kernel:
  -Calculate the Euclidean distance between each data point and each centroid.
  -Determine the closest centroid for each data point based on the computed distances.
  -Assign data points to clusters based on the closest centroid.
  -Store the distances and cluster assignments for each data point.

- Synchronize device threads to ensure all computations are completed before proceeding.

- Print and measure the execution time for different block sizes to evaluate performance.

- Compute the speedup achieved by parallel execution

- Free allocated host and device memory using CUDA memory management functions.

Advantages of hpc for Kmeans clustering for MRI images

- Faster Processing Time
- Improved Scalability
- Memory Capacity
- Resource Management

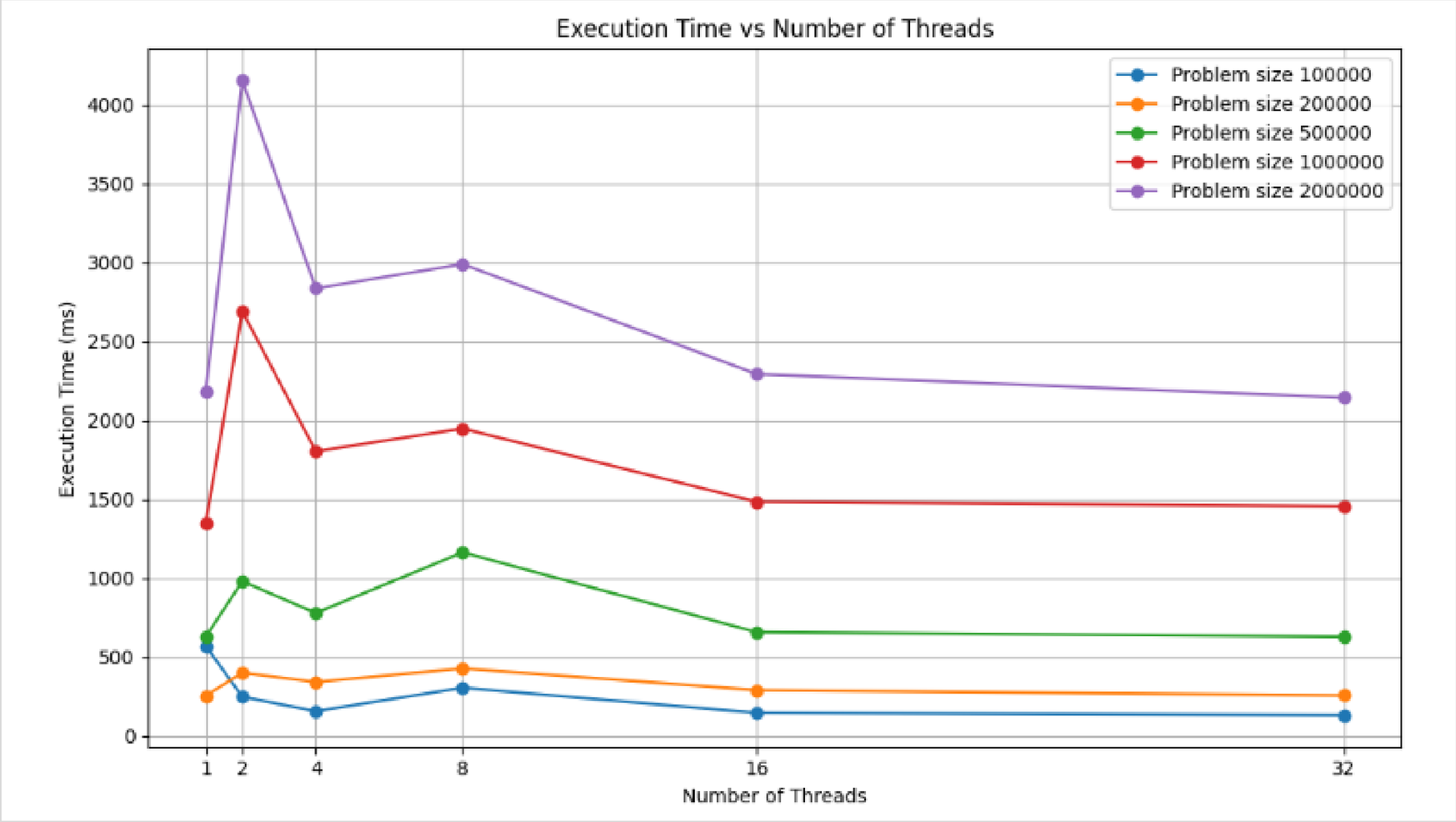
Advantages of Parallelism in K-means clustering of MRI images:

- **Reduced Processing Time:** MRI images are high-dimensional and voluminous. Parallelization distributes the workload across multiple threads significantly speeding up the process. This is especially crucial for analyzing large datasets or performing real-time analysis.

- **Improved Scalability:** Parallelization allows you to leverage the computing power of multi-core GPUs. As the number of cores or processing units increases, the processing time scales efficiently, making K-means clustering feasible for even bigger MRI datasets.

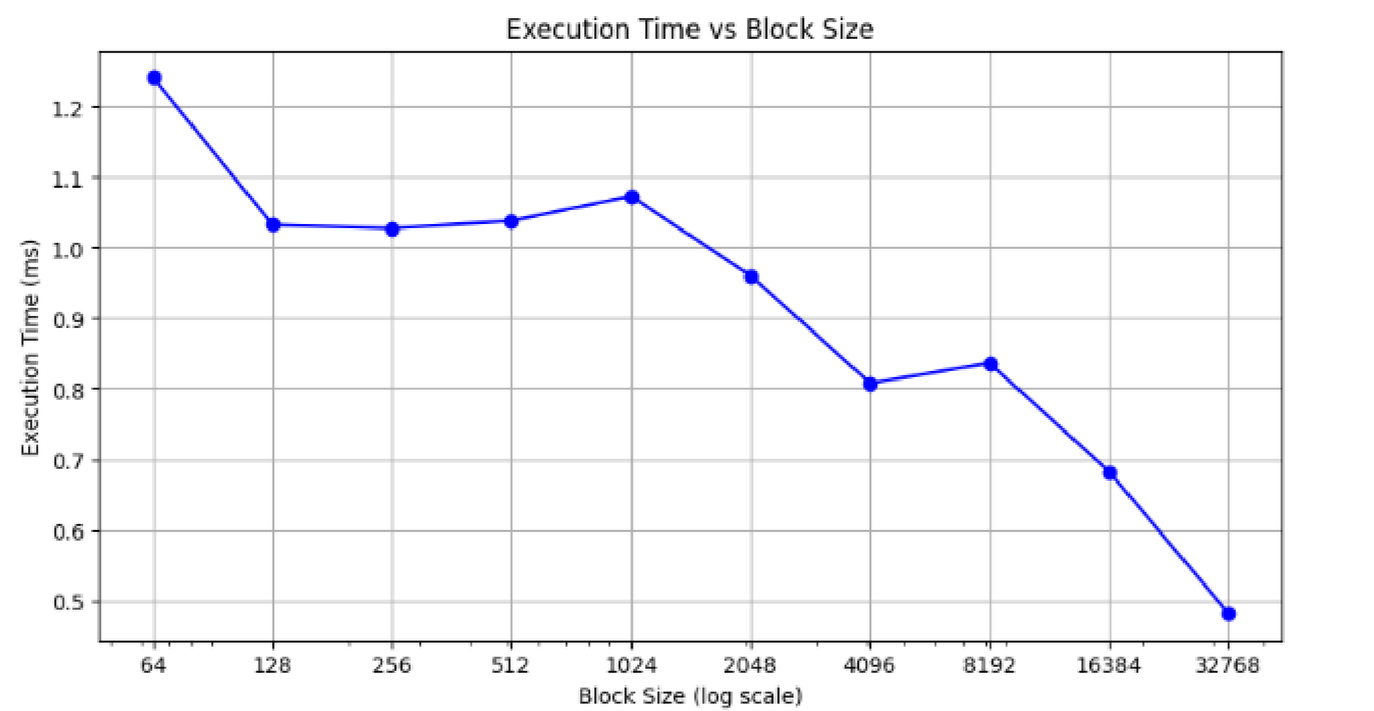
- **Faster Iteration:** K-means clustering is iterative, involving multiple assignments of data points to clusters and recalculating centroids. Parallelization can accelerate these calculations, leading to quicker convergence on the final clusters

-RESULTS (OPEN MP) :



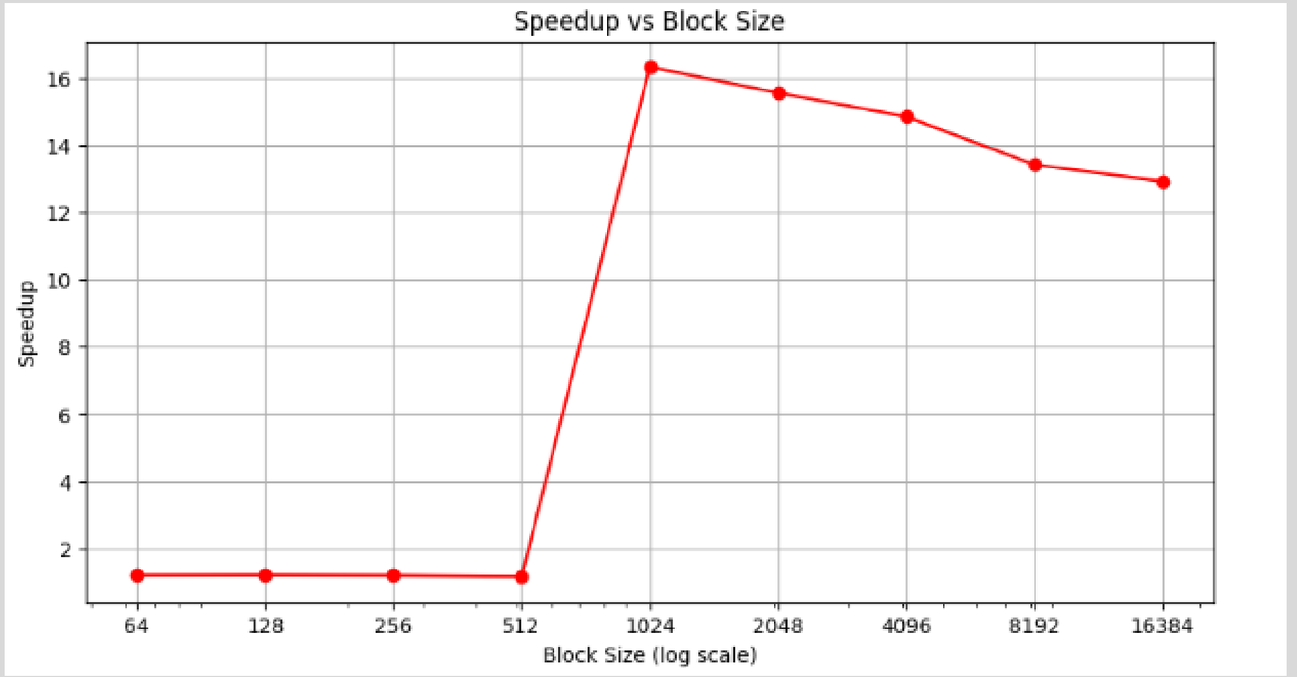
Configuration	Speedup
Problem size 1, Threads 1	0.635566
Problem size 1, Threads 2	0.744833
Problem size 1, Threads 4	0.594922
Problem size 1, Threads 8	0.875149
Problem size 1, Threads 16	0.989456
Problem size 2, Threads 1	0.640268
Problem size 2, Threads 2	0.805518
Problem size 2, Threads 4	0.539463
Problem size 2, Threads 8	0.954125
Problem size 2, Threads 16	0.997086
Problem size 3, Threads 1	0.501791
Problem size 3, Threads 2	0.748541
Problem size 3, Threads 4	0.693158
Problem size 3, Threads 8	0.909701
Problem size 3, Threads 16	0.927349
Problem size 4, Threads 1	0.526288
Problem size 4, Threads 2	0.77067
Problem size 4, Threads 4	0.731132
Problem size 4, Threads 8	0.952818
Problem size 4, Threads 16	1.01959

-RESULTS:



Speedup for different block sizes

Block Size	Speedup
64	1.20257
128	1.4083
256	3.69594
512	5.85721
1024	16.326
2048	15.5539
4096	14.8458
8192	13.4187
16384	12.9227



Speedup for different block sizes:

Block Size	Speedup
64	1.20257
128	1.2083
256	1.19594
512	1.15721
1024	16.326
2048	15.5539
4096	14.8458
8192	13.4187
16384	12.9227

Execution times for different block sizes:

Block Size	Execution Time (ms)
64	1.24317
128	1.03491
256	1.02704
512	1.01478
1024	1.02515
2048	0.80912
4096	0.10416
8192	0.08512
16384	0.09656
32768	0.079808

Speedup for different block sizes

Block Size	Speedup
64	1.15677
128	1.19714
256	1.18081
512	1.1993
1024	15.8433
2048	15.1578
4096	14.3719
8192	13.4442
16384	12.8394

-DRAWBACKS:

- Since our code has a large number of arrays and different variables to be copied to and fro, we need to transfer data everytime a new variable/memory space is introduced, or is modified.

SOLUTION:

- To overcome this issue, we can create the concept of Unified memory. Unified Memory creates a pool of managed memory, where each allocation from this memory pool is accessible on both the host and device with the same memory address.

- OBSERVATIONS:

- Speed up increases consistently upto a point with respect to block size, after which speedup decreases due to factors such as diminishing returns in parallelism and limitations imposed by Amdahl's Law.

- Execution time decreases consistently with increase in block size.

Speedup for different block sizes

Block Size	Speedup
64	1.20257
128	1.4083
256	3.69594
512	5.85721
1024	16.326
2048	15.5539
4096	14.8458
8192	13.4187
16384	12.9227

REFERENCES:

*Papers referred:

-Accelerating K-Means Clustering with Parallel Implementations and GPU computing
(http://nucsrl.coe.neu.edu/sites/nucsrl.coe.neu.edu/files/janki/HPEC'15-Janki.pdf)
-Chest Disease Image Classification Based on Spectral Clustering Algorithm
https://www.researchgate.net/publication/371762710_Chest_Disease_Image_Classification_Based_on_Spectral_Clustering_Algorithm

-GITHUB REPO:
https://github.com/pikulkarni7/K-Means-using-CUDA

CHALLENGES:

-Initializing centroids
-Determining number of threads