K-Means Clustering Project Report

1. Introduction

1.1 Project Overview

This project implements and compares sequential and CUDA-based parallel K-means clustering algorithms to evaluate performance improvements from GPU acceleration.

1.2 Background on K-Means Clustering

K-means is an unsupervised learning algorithm that partitions data into k clusters by minimizing within-cluster variance.

Algorithm Steps:

1. Initialize k centroids randomly.
2. Assign each data point to the nearest centroid (Euclidean distance).
3. Recompute centroids as the mean of assigned points.
4. Repeat until convergence.

Limitations:

- Sensitive to initial centroids.
- Assumes spherical clusters.
- Requires predefined k.

1.3 Motivation

K-means is computationally expensive for large datasets. GPU parallelization using CUDA can drastically reduce processing time, enabling efficient large-scale data analysis.

2. Methodology

2.1 Data Description

Synthetic datasets with varying sizes (1M–10M points) and dimensions (5–30) were used to test scalability.

2.2 Implementation Details

2.2.1 Sequential K-Means (C++)

- Random centroid initialization.
- Iterative assignment and centroid updates.
- Sum of Squared Errors (SSE) for convergence.

2.2.2 CUDA Parallel K-Means

- **Parallelized Steps:**
  - Distance calculations (Euclidean), cluster assignment, and SSE calculation.
  - New Centroid Selection by taking the mean of the existing data points.
- **Optimizations:**
  - GPU memory management.
  - Shared memory for faster data access.
- **Hardware:** NVIDIA A100 GPU.

2.3 Setup

- Use Python code to generate test data, Use Command **python generate_data N D**
- Once the Data is generated, use the makefile to compile the sequential and CUDA code
- run the clustering using command **./kmeans_** path_to_bin_file N D K**

3. Results

3.1 Performance Comparison

Table 1: Performance Comparison

Dataset (N, D)	K	Sequential (ms)	CUDA (ms)	Speedup
1M points, 5D	20	1916	91	21.05x
10M points, 10D	10	32303	1625	19.88x
5M points, 30D	3	9136	2414	3.78x

Key Observations:

- Highest speedup (∼21x) for large, low-dimensional data.
- Lower speedup (∼3.78x) for high-dimensional data due to increased computation.

3.2 Accuracy Analysis

Both implementations converged to almost similar SSE values, confirming that CUDA does not compromise accuracy.

3.3 Scalability Analysis

CUDA scales efficiently with larger datasets, but speedup depends on dimensionality and k.

4. Analysis & Conclusion

4.1 Amdahl's Law Analysis

- The speedup depends on the sequential part of the code. (First centroid selection and Subsequent centroid selection)

4.2 Discussion

- **CUDA outperforms CPU** significantly for large datasets.

5. Conclusion & Future Work

- **Conclusion:** CUDA accelerates K-means effectively (up to 21x speedup).
- **Future Work:**
  - Further parallelization of the sequential part of the code in CUDA using **OpenMP** to get much better results.
  - Test on real-world datasets.

5. Application is Real World

- **LLM:** Used for performing spherical k-means clustering and cosine similarity calculation on the vast amount of text-based LLM.
- Unsupervised Machine Learning Classification-based Predictions