

CMPXXX

Big Assignment

(Kmeans Clustering)

Team x

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Sec** | **B.N** | **Code** |
| Basma Elhoseny | 1 | 16 | 9202381 |
| Sara Mohamed Hossam | 1 | 29 | 9202618 |

Table of Contents

[Sklearn Implementation 3](#_Toc166249348)

[Version (1): 3](#_Toc166249349)

[CPU C Implementation 4](#_Toc166249350)

[Version (1): 4](#_Toc166249351)

[Design: 4](#_Toc166249352)

[Profiling: 4](#_Toc166249353)

[Insights: 4](#_Toc166249354)

[GPU C Implementation 5](#_Toc166249355)

[Version (1) [Very Simple Inefficient Approach]: 5](#_Toc166249356)

[Design: 5](#_Toc166249357)

[Improvements Ideas: 5](#_Toc166249358)

[Profiling: 5](#_Toc166249359)

[Insights: 5](#_Toc166249360)

[Progress Table 6](#_Toc166249361)

# Sklearn Implementation

## Version (1):

In this Version

# CPU C Implementation

## Version (1):

A screen shot of a computer program

Description automatically generated

## Design:

## Profiling:

## Insights:

# GPU C Implementation

## Version (1) [Very Simple Inefficient Approach]:

In this Version we have mainly focused on implementation of the parallel version of the algorithm without logical or runtime error (taking not into consideration any possible optimization) Following Golden Rule accumulative baby steps move mountains 😍

A screen shot of a computer code

Description automatically generated

## Design:

A screenshot of a computer screen

Description automatically generated

In this Version

intilize\_centroids() is CPU Version

Kernel(1) assign\_data\_points\_to\_centroids():

Each Thread is Responsible for 1 Pixel to get nearest centroid to it

d\_data\_points & d\_centroids are read from Global Memory

d\_cluster\_assignment is written by each thread to the global Memory.

[TODO] Check Col leasing Memory Accesses

Kernel(2) update\_cluster\_centroids():

shared\_data\_points in shared Memory for each block.

Each thread is responsible for loading 1 datapoint (RGB) to the shared memory.

shared\_cluster\_assignment in shared Memory for each block.

each thread loads the cluster assignment to shared memory

Thread Zero only [TODO] Fix to be Dynamic

1. Defines data\_point\_sum[K\_max\*D] = {0}; // sum of data points for each cluster
2. int cluster\_size[K\_max] = {0}

loops over datapoints in shared\_data\_points(shared) and add its sum to the coresspoding cluter sum in data\_point\_sum(reg) & data\_point\_sum(reg)

when Done it add data\_point\_sum to d\_centroids (atomic sum)

and cluster\_size to d\_cluster\_sizes (atomic sum)

In CPU we divide sum centroids / cluster size

Check Convergence

clutser\_image()

get colors for clusters

loop over each pixel and assign it color of its final cluster

shared\_cluster\_assignment is shared Memory for each block.

## Improvements Ideas:

xx

## Profiling:

xx

## Insights:

Although this version is very inefficient with redundant memory copies and global memory access still it is 18x faster than CPU 😉. This is great evidence that this problem is very efficient parallelizable problem more optimizations will let us reach incredible improvements in speed 🚀.

|  |  |  |  |
| --- | --- | --- | --- |
|  | V1  gpu\_3.cu | V2(Shared Memory for centroids in K1)  **Added Shared Memory Centroids for K1**  gpu\_3\_1.cu | gpu\_3\_2.cu |
|  | 0.047000 | 0.032000 |  |
| Converge after | 17 | 12 | 17 |
| update\_cluster\_centroids | 82.99% 12.738ms  17 749.27us | 82.39% 9.0448ms  12 753.73us | 68.19% 5.0311ms  17 295.95us |
| assign\_data\_points\_to\_centroids | 8.24% 1.2651ms  17 74.417us | 8.59% 943.18us  12 78.598us | 18.17% 1.3404ms  17 78.847us |
| CUDA memcpy HtoD | 6.86% 1.0531ms  18 58.503us | 7.19% 789.23us  13 60.709us | 10.97% 809.10us  18 44.949us |
| CUDA memcpy DtoH | 1.87% 287.40us  35 8.2110us | 1 1.80% 197.35us  25 7.8930us | 2.60% 191.94us  35 5.4840us |

# Progress Table

The Results of this Table are based on same input image ./tests/image\_3.png with K=5 :D to just sense our improvements.

|  |  |
| --- | --- |
| **Version** | **Time** |
| **CPU C Version** | 0.388 |
| **GPU Version (1)** | 0.02100 |
|  |  |
|  |  |