HPC Programming Project - 2024/2025

Pham Gia Phuc

November 2024

1 Introduction

This project focuses on clustering techniques using two methods on the GPU for data clustering, based on centroids:

- **Classical k-means** (KM), referred to hereafter as `kMeans`.
- **Fuzzy k-means** (FCM) (also known as **Fuzzy C-Means** or **Soft k-Means**), referred to hereafter as `FCM`.

2 Implementation

To achieve the goal of this study, I have implemented the work in 4 step:

- **Step 1**: Initial implementations of KMeans (KM) and Fuzzy C-Means (FCM) were created for the CPU.
- **Step 2**: Transitioned these implementations to the GPU using Numba, initially without shared memory optimization.
- **Step 3**: Optimized further by utilizing shared memory and enhancing parallelization.
- **Step 4**: Conducted multiple rounds of refinement to reduce runtime and computational complexity.

Furthermore, this study uses a CUDA kernel provided by Google Colaboratory with 2 images being used as entry data for quantization as shown in Figure 1.

Figure 1: Sample image data.

2.1 CPU implementation

2.1.1 Problems

The first implementation is extremely slow performance due to the formula of calculating distance using Euclidean distance such that it calculate the distance between each pixel of the image to the centroid once per loop, taking a lot of computational power.
On the other hand, the FCM's membership matrix update was particularly costly due to its exponentiation and multiple distance computations.

2.1.2 Solution

Vectorized operations are being used to enhance the CPU performance, significantly reduce running time of the implementation.

2.2 GPU transitioning without shared memory

2.2.1 Problems

When transitioning from CPU to GPU, there are a bunch of issues with thread and block size that I have encountered, especially around dividing and handling empty clusters.
In addition, there are also issues with Not a number (NaN) and Infinity (Inf) values (for KM) and distance calculation in membership matrix (for FCM) which lead to only one color in quantized images.

2.2.2 Solution

Adjust thread and block and handling empty clusters, NaN/Inf values, as well as implemented a k-means++ like initialization strategy instead of random initialization

Moreover, properly normalized and added better numerical stability with maximum distance threshold for the membership values help solve the issues of empty/NAN/Inf values.

2.3 GPU transitioning with shared memory

2.3.1 Problems

When dealing with shared memory and parallelization, at first, the shared memory did not work efficiently which caused suboptimal performance.
Also, the atomic operations potentially led to bottlenecks, and thread divergence slowed down execution.

2.3.2 Solution

Refined shared memory management and thread synchronization and improving efficiency in centroid calculations, ensure parallelism of threads.

2.4 Optimization

2.4.1 Problems

Thread divergence, underutilized threads, and excessive synchronization are the main factors who reduced the overall performance of the implementation.
Similar as before, the use of atomic operations potentially slows down parallel processing

2.4.2 Solution

Limit the use of cuda.syncthreads() to essential points, optimized the number of threads, and minimized atomic operations.

3 Results

3.1 Performance metrics

3.1.1 Computation time

Table 1 demonstrates the computation time of each phrase of testing.

	KMeans	**Fuzzy C-Means**
CPU	30+	50+
GPU without shared memory	10+	18+
GPU with shared memory	2.8+	3.8+
GPU with optimization	0.4+	1.4+

Table 1: Relative runtime comparison with a 640x800 pixels image (in seconds)

The implementations with GPU reduced running time significantly compare to running on CPU.

3.1.2 Cost function value

Because of too focusing on adjusting the computational time, I have yet finish implementing the functions to calculate the cost function value.

3.2 Quantization Accuracy

The two figures 2 and 3 illustrate the quantization accuracy of K-Means clustering and Fuzzy C-Means clustering methods. Both methods give the similar result for the work of clustering.

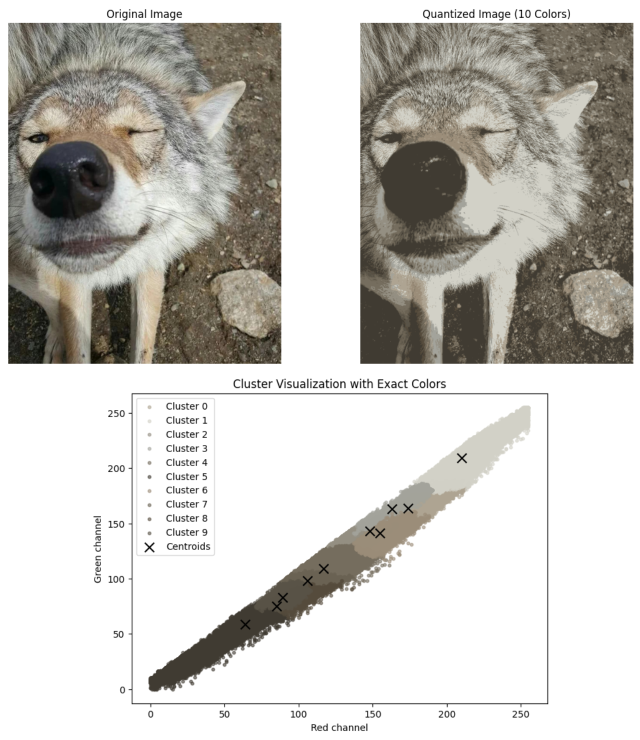

Figure 2: Sample result with KMeans

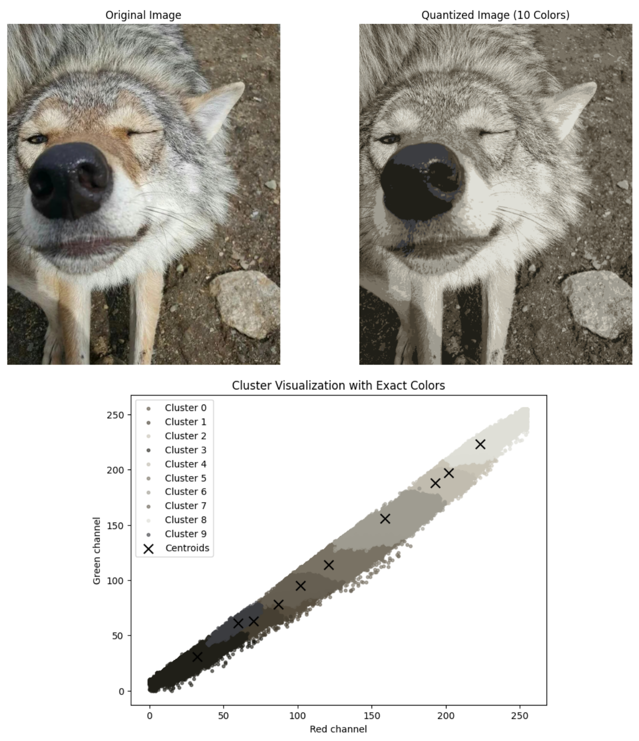

Figure 3: Sample result with Fuzzy C-Means

4 Conclusion

In this study I have made the GPU implementations of KMeans and FCM using CUDA, with iterative optimizations through six main tests. Each phase refined vectorization, parallelization, and memory management to try achieving the goal of reducing computational time as well as value of cost function.
Although the results are quite decent, there might be still much room for improvement of this study's objective.