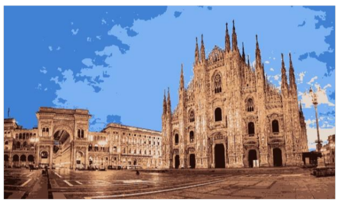
Parallel Kmenas Images Compressor

Leonardo Ignazio Pagliochini
Francesco Rosnati

June 17, 2024



Abstract

In this report, we present a parallel implementation of the K-means algorithm for image compression. The algorithm is implemented in C++ using OpenMP, MPI and CUDA to parallelize the computation. The report describes the algorithm, the implementation, and the results of the experiments. The results show that the parallel implementations are faster than the sequential one, and that the CUDA implementation is the fastest among the three parallel versions. The report also discusses the limitations of the parallel implementation and possible future improvements.

1 Introduction

K-means is a widely used technique in data analysis and machine learning for grouping similar data points together. It is based on the heuristic that data points that are close to each other in the feature space are likely to belong to the same group. The algorithm works by iteratively assigning data points to the nearest cluster center and then updating the cluster centers based on the new assignments. The algorithm converges when the cluster centers no longer change significantly between iterations. Differently from other clasterin algorithms, K-means needs to know the number of clusters in advance. This is a limitation of the algorithm, but it is also one of its strengths, as it allows the user to specify the number of clusters based on domain knowledge or other criteria. In particular in this project we are interested in the application of the K-means algorithm to image compression to reduce the number of colors in an image. The idea is to represent the image with a smaller number of colors, while preserving the overall structure of the image. This can be useful for reducing the size of the image file, for example to speed up the loading of web pages or to save storage space.

2 The Kmeans Algorithm

In this section we describe the K-means algorithm in detail. To understand the algorithm, we first need to define some terms.
Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of data points in a d-dimensional space, where $x_i \in R^d$.
The goal of the K-means algorithm is to partition the data points into k clusters, where k is a user-specified parameter. The algorithm works as follows:

1. Initialize the cluster centers $\mu_1, \mu_2, \ldots, \mu_k$ randomly or using some other heuristic.
2. Assign each data point x_i to the nearest cluster center μ_j.
3. Update the cluster centers μ_j by taking the mean of all data points assigned to cluster j.
4. Repeat steps 2 and 3 until the cluster centers no longer change significantly between iterations.

Here we present the pseudocode of the K-means algorithm.

Algorithm 1 K-means algorithm

```
def Kmeans(X, k):
# Initialize the cluster centers
for i in range(k):
  mu_i = random_point()

# Main loop
while True:
  # Assign data points to clusters
  for i in range(n):
    j = argmin_j ||x_i - mu_j||
    C_j = C_j + {x_i}

  # Update cluster centers
  for j in range(k):
    mu_j = (1/|C_j|) * sum(x for x in C_j)

  # Check for convergence
  if sum(||mu_j - mu_j_old|| for j in range(k)) < epsilon:
    break
```

Where:

- X is the set of data points
- k is the number of clusters
- n is the number of data points
- μ_i is the cluster center for cluster i
- C_j is the set of data points assigned to cluster j

- $||x_i - mu_j||$ is the Euclidean distance between data point x_i and cluster center μ_j
- $argmin_j ||x_i - mu_j||$ is the index of the cluster center closest to data point x_i
- $epsilon$ is a small positive number that determines the convergence threshold

2.1 The Image Compression