K-Means Algorithm

Implementation of sequential and parallel version using OpenMp

Marco Trambusti - 7135350

23-11-2024

Introduction

Purpose of the Assignement

- Implement the K-Means algorithm in C++ in both sequential and parallelized versions using OpenMP.
- Dataset: 300147 values related to age and total spending of supermarket customers.
- Performance comparison on a machine with an Intel Core i7-6700K processor (4 cores).

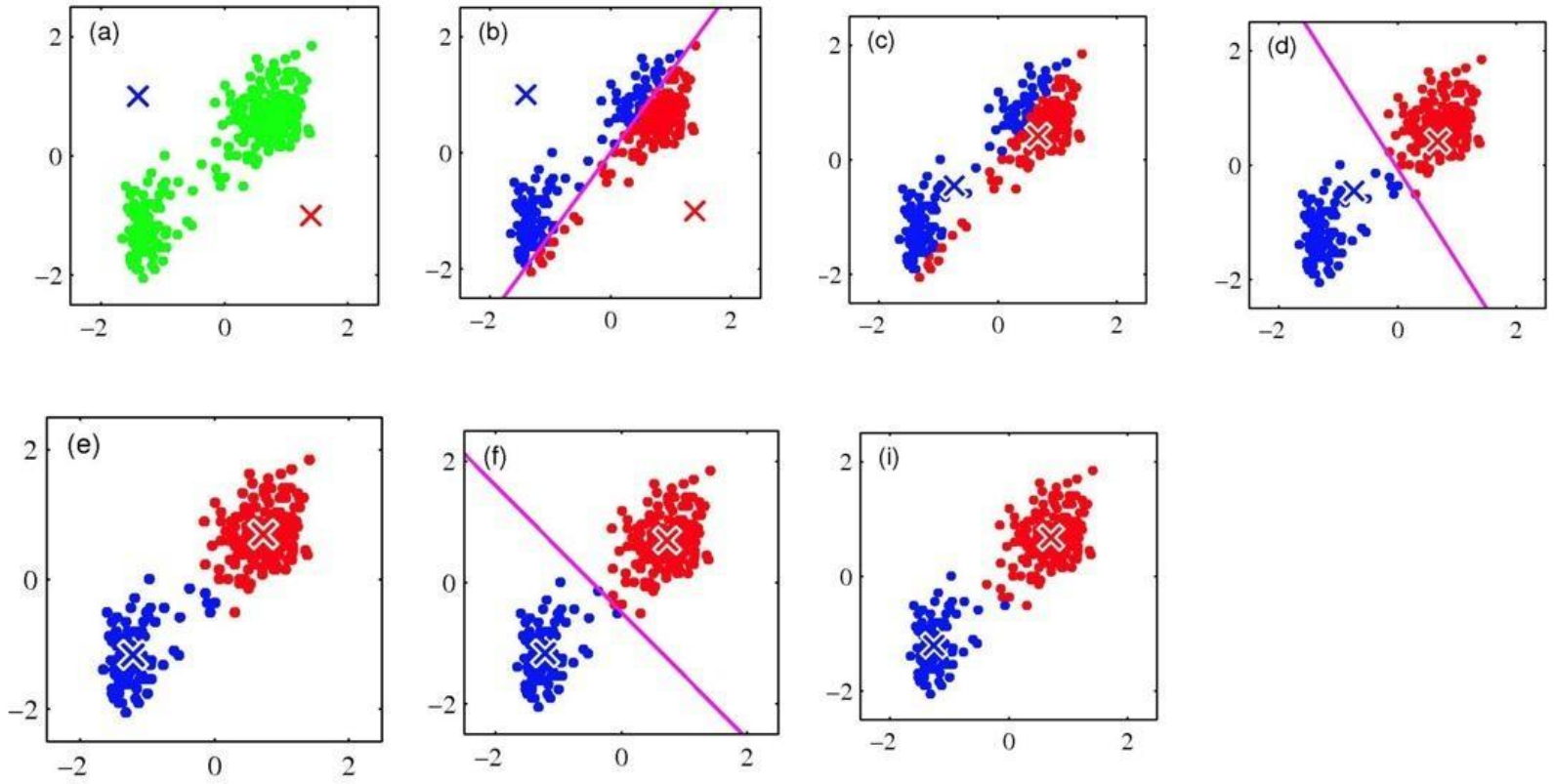
K-Means Algorithm

1. **Initialization**: Random selection of K initial cluster centers.

2. **Cluster Assignment:** Each data point is assigned to the nearest centroid.

3. **Centroid Update:** Centroids are recalculated as the mean of the points assigned to each cluster.

4. **Iteration:** The assignment and update steps are repeated until the centroids no longer change significantly or a maximum number of iterations is reached.



Esempio di K-Means

Algorithm Parameters

- Number of clusters (K)
- Initialization of centroids
- Choice of distance metric (squared Euclidean distance)
- Maximum number of iterations

Sequential Implementation

It was chosen C++ for the sequential implementation and OpenMP for the parallel implementation.
In this program, 2D points points are readed from a csv file and clusters are initialized randomly;

The points in the dataset are represented using a struct **Point**, composed by:

- x, y coordinates (double)
- cluster (int);
- minDist (double)
- distance(Point p) (method)

Initially, the point does not belong to any cluster, and was arbitrarily set to -1.
As a result, minDist was set to the maximum possible value.

Sequential Implementation

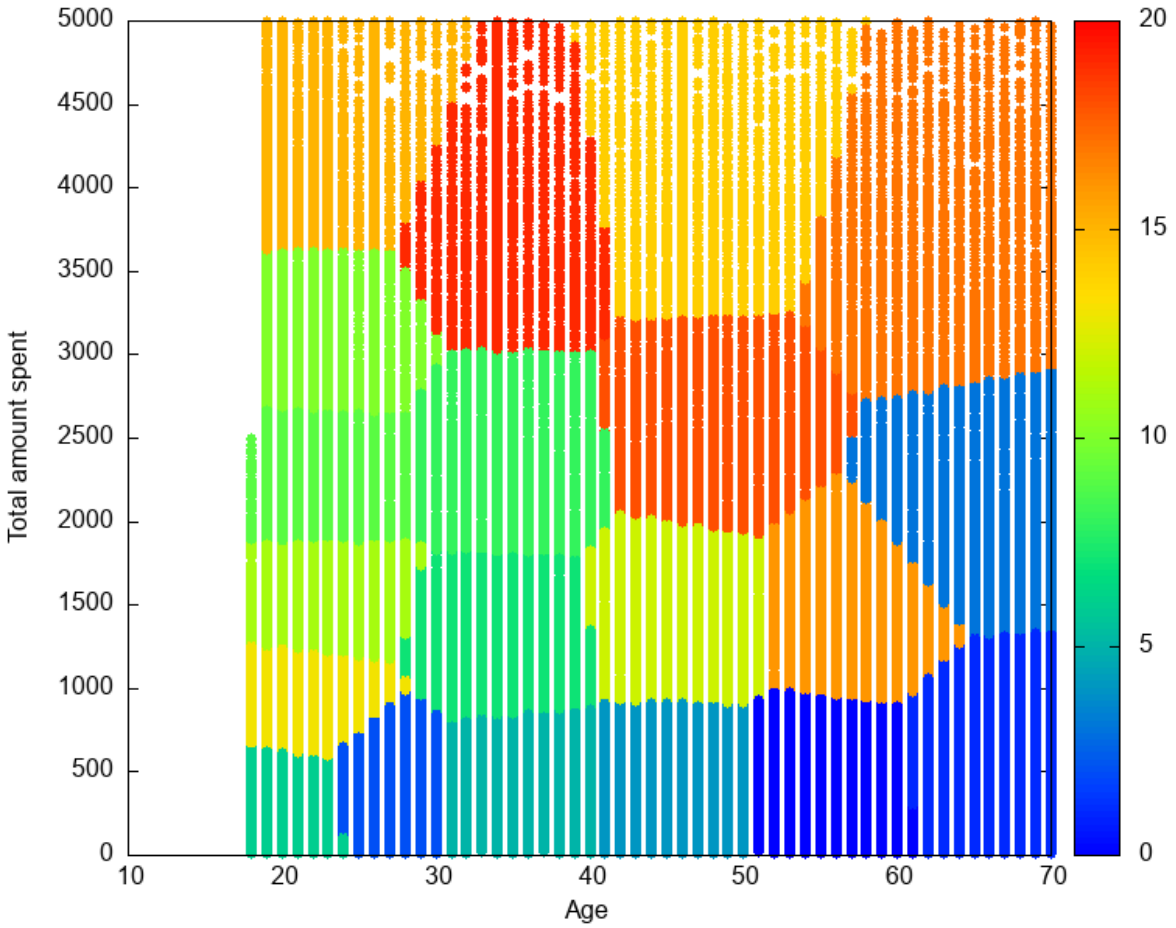
Kmeans method: Cluster assigment part

```
1  void kmeans(std::vector<Point>& points, std::vector<Point> centroids,
2      int k, int epochs) {
3      for (int epoch = 0; epoch < epochs; ++epoch) {
4          int nPoints[k]= {0};
5          double sumX[k] = {0.0};
6          double sumY[k] = {0.0};
7          for (auto& point : points) {
8              for (int i = 0; i < k; ++i) {
9                  double dist = point.distance(centroids[i]);
10                 if (dist < point.minDist) {
11                     point.minDist = dist;
12                     point.cluster = i;
13                 }
14             }
15             //append data to centroids
16             int clusterId = point.cluster;
17             nPoints[clusterId]++;
18             sumX[clusterId] += point.x;
19             sumY[clusterId] += point.y;
20             //reset distance
21             point.minDist = std::numeric_limits<double>::max();
22         }
23         ...
```

Sequential Implementation

Kmeans method: Centroid Update Code

```
1       for (int i = 0; i < k; ++i) {
2           if (nPoints[i] > 0) {
3               double newX = sumX[i] / nPoints[i];
4               double newY = sumY[i] / nPoints[i];
5               centroids[i].x = newX;
6               centroids[i].y = newY;
7           }
8       }
```



Example of results plotted using GnuPlotted

Parallel Implementation

Cluster Assignment

- Cluster assignment step is independent for each data point,
  - ideal candidate for parallelization.
- Using OpenMP, it is possible to distribute the calculation parallelizing the outer loop.
- Updating the sum of coordinates and the number of points relative to the cluster, adds complexity due to data sharing,
  - Optimized using reductions

```
1 #pragma omp parallel default(none) shared(points, centroids,
2 nPoints, sumX, sumY,k) if(omp_get_max_threads() > 1)
3 {
4     // Assign points to the nearest centroid
5     #pragma omp for reduction(+:nPoints,sumX,sumY)
6     for (auto& point : points) {
7         for (int i = 0; i < k; ++i) {
8             ...
```

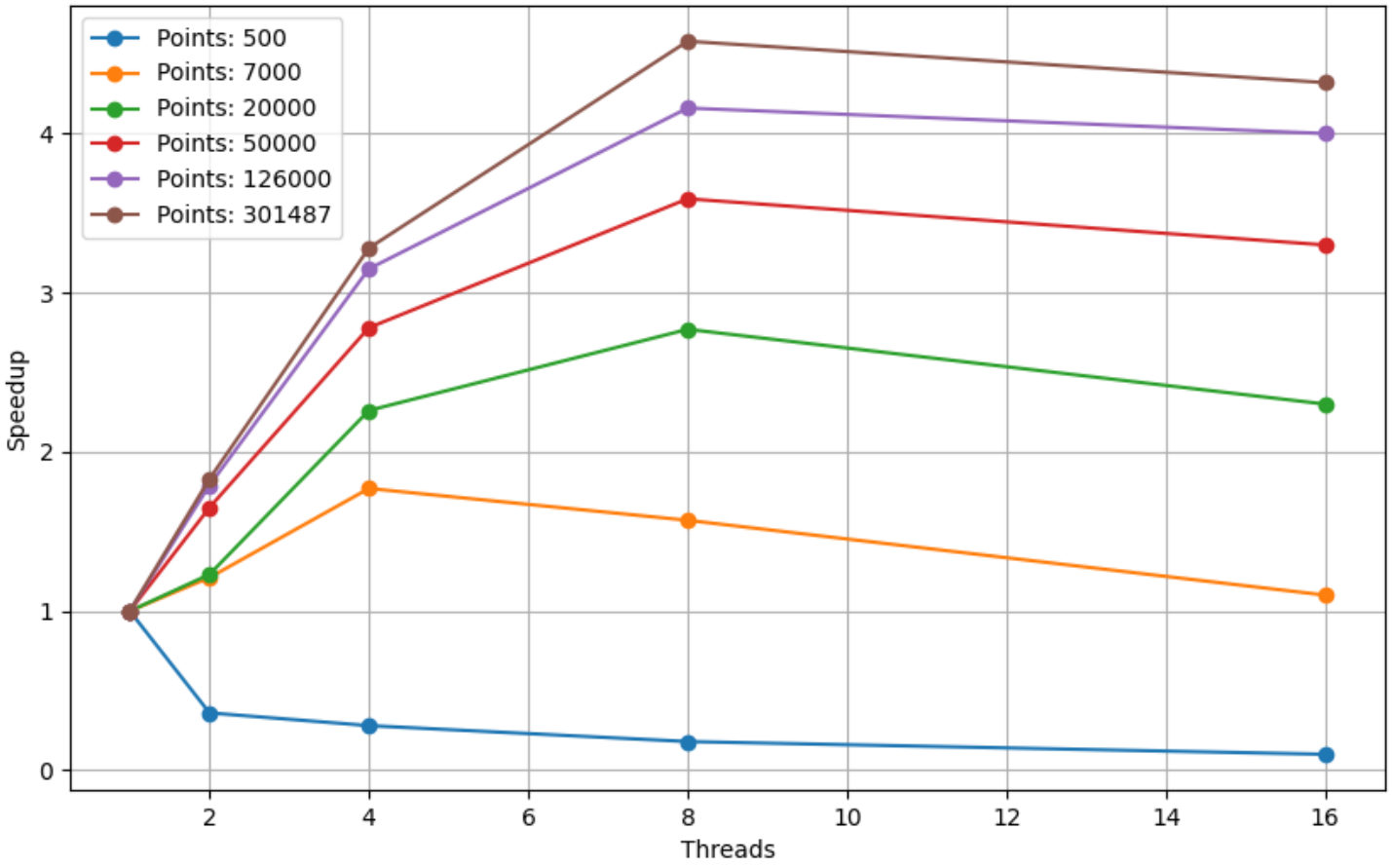
Parallel Implementation

Centroids Update

```
1 // Update centroids
2 #pragma omp for
3 for (int i = 0; i < k; ++i) {
4     if (nPoints[i] > 0) {
5         ...
6     }
7 }
```

Results and Performance Evaluation

Speedup: Ratio of sequential to parallel execution time.



Conclusions

1. Parallelizing the K-Means algorithm proved effective, especially with larger datasets.

2. OpenMP simplified implementation and performance optimization.

3. The larger the number of points, the more effectively threads can be utilized up to a certain point.