UNIVERSITÀ DEGLI STUDI FIRENZE

Meanshift Implementatoin

Parallel Computing 9CFU

Claudio Pagnini

Introduction

- MeanShift is another popular clustering algorithm used for **density estimation** and **finding modes** in a dataset.
- Unlike K-Means, MeanShift does not require specifying the number of clusters (**K**) beforehand.
- Instead, it automatically determines the number of clusters by finding regions of high data density, which are called **modes**.
- As a result, data points within the same mode gravitate towards a single cluster.

Introduction

Given a set of observations in a 2D dimensional space, for each data point x_i, we find its neighbors x_j within a specified range λ using a flat kernel K(x). The flat kernel is defined as:

$$\begin{cases} 1 & \text{if } \|x\| \leq \lambda \\ 0 & \text{if } \|x\| > \lambda \end{cases}$$

Introduction

Update Step: Next, we calculate the updated position x' for each data point:

$$x' = \frac{\sum_{x_j} K(\|x_i - x_j\|) \cdot x_j}{\sum_{x_j} K(\|x_i - x_j\|)}$$

We update the position of each centroid c_j based on the mean of the data points assigned to its cluster:

$$\text{New centroid position } c_j = \frac{1}{n_j} \sum_{x_i \text{ assigned to cluster } j} x_i$$

Where n_j is the number of data points assigned to cluster j.

UNIVERSITÀ DEGLI STUDI FIRENZE

Proposed approach

In this work we propose two different implementations of the algorithm:

1. A sequential version with C++
2. A parallel version with OpenMP using AoS and SoA architectures

UNIVERSITÀ DEGLI STUDI FIRENZE

Proposed approach

”AoS” and ”SoA” are two different ways of organizing data in memory, particularly when dealing with collections of related elements

AoS (Array of Structures) groups related data into an array of structures, while **SoA** (Structure of Arrays) organizes data into separate arrays based on attributes. AoS is intuitive and easy to use, while SoA can offer performance benefits in certain scenarios, especially for large datasets and optimized memory access.

UNIVERSITÀ DEGLI STUDI FIRENZE

Sequential Implementation

Core

```
new_points<-original_points
REPEAT until MAX_ITERATION
    MeanShift(original_points,new_points, lambda)
```

Shift point

```
weight<-0
REPEAT for each points
    REPEAT until point_size
        distance<-Euclidean_Distance(point)
        if(distance<lambda)
            new_point <- new_point + original_point
            weight <- weight +1
    if(weight!=0)
        new_point <- new_point / weight
    else
        new_point<-point
```

OpenMP

The OpenMP API allow to transform the sequential version into a parallel one with several directives

Shift Point OpenMp

```
weight<-0
#pragma omp parallel firstprivate(points_size)
#pragma  omp for schedule(static)
REPEAT for each points
    REPEAT until point_size
        distance<-Euclidean_Distance(point)
        if(distance<lambda)
            new_point <- new_point + original_point
            weight <- weight +1
    if(weight!=0)
        new_point <- new_point / weight
    else
        new_point<-point
```

We then checked for non-zero shifted points

Non zero shifted points

```
REPEAT for each points
        if(shifted_points !=0)
            non-zeroPoints<-point
```

UNIVERSITÀ DEGLI STUDI FIRENZE

Experimental Results

The metric used to compare the performances of the sequential algorithm with the OpenMP and Cuda implementation is the speedup, computed as:

$$S = \frac{t_S}{t_P}$$

where t_S and t_P are respectively the execution time of the sequential and the parallel implementation
The tests have been executed on a Laptdop that do not allow logical cores with:

- OS: Windows 11 Pro
- CPU: AMD Ryzen 5 4500U CPU @2.38GHz
- GPU: GEForce GTX 1650 Mobile / 4GB with CUDA 11.1

UNIVERSITÀ DEGLI STUDI FIRENZE

Experimental Results

Parameter used for each experiment has been:

- $\lambda = 1$
- MAXITERATION = 10

The datasets used to evaluate the different implementations have been generated with the make_blob() function of sklearn.datasets . They are gaussian distributions with 5 centers and standard deviation equal to 0.5 and are composed by respectively 100, 1000, 10000, 100000 and 10000002D points.

UNIVERSITÀ DEGLI STUDI FIRENZE

Performance Evaluation - OpenMp

To evaluate the performances of the OpenMP implementation, it has been executed on each dataset with the same number of threads comparing AoS vs SoA architectures. The results for the 100, 1000, 10000 and 100000 dataset are shown respectively in Table 1 and 2.

UNIVERSITÀ DEGLI STUDI FIRENZE

Performance Evaluation - OpenMp

Dim	Seq Time	AoS Time	Speedup
100	0.0016 s	0.0021	0.761
1000	1.178 s	0.473 s	2.490
10000	112.966s	40.072 s	2.81
100000	11175.633 s	4031.578 s	2.77

Table 1. Speedup for 100, 1000, 10000, 100000 points using AoS arhchitecture

Dim	Seq Time	SoA Time	Speedup
100	0.0016 s	0.0017	0.941
1000	1.178 s	0.3990 s	2.952
10000	112.966s	35.188 s	3.210
100000	11175.633 s	3467.315 s	3.223

Table 2. Speedup for 100, 1000, 10000, 100000 points using AoS arhchitecture

UNIVERSITÀ DEGLI STUDI FIRENZE

Conclusion

After conducting a comprehensive analysis, the best results for each dataset dimension and implementation have been compared. The following table presents the findings, demonstrating that the OpenMP SoA algorithm outperforms both the sequential and OpenMP AoS implementations, with the exception of the dataset containing 100 points, where the sequential approach performs better.

Dim	Sequential	OpenMP AoS	AoS SpeedUp	OpenMP SoA	SoA Speedup
100	0.0016 s	0.0021	0.761	0.0017	**0.941**
1000	1.178 s	0.473 s	2.490	0.3990 s	**2.952**
10000	112.966s	40.072 s	2.81	35.188 s	**3.210**
100000	11175.633 s	4031.578 s	2.77	3467.315 s	**3.223**