K-Means Study

# Objective

The objective of this study is to fulfill the requirements of the CS-770 Big Data Analysis class and further gain more understanding and experience in parallelizing serial algorithms in C.

# Work

For the purpose of this CS-770 Big Data Analysis class project I performed the following:

* chose K-Means clustering method
* used OpenMP to parallelize ([k-means-omp.c](https://github.com/niko-todorov/OMP/blob/main/k-means-omp.c)) the algorithm’s serial ([k-means-ser.c](https://github.com/niko-todorov/OMP/blob/main/k-means-ser.c)) version in C
* used three input files the data of which was generally grouped in 4 major clusters with
  + [10k](https://github.com/niko-todorov/OMP/blob/main/data/dataset-10000.txt) 3D data points,
  + [100k](https://github.com/niko-todorov/OMP/blob/main/data/dataset-100000.txt) 3D data points, and
  + [1m](https://github.com/niko-todorov/OMP/blob/main/data/dataset-1000000.txt) 3D data points
* ran the OpenMP executable on
  + 4 threads,
  + 8 threads, and
  + 16 threads to find the best optimization for my laptop
* compared the 3 multi-threaded OpenMP vs single-threaded serial execution results
* wrote a short R script using **rgl** ([3dscatter.R](https://github.com/niko-todorov/OMP/blob/main/3dscatter.R)) for interactive 3D scatterplot cluster visualization
* visualized the 10k and 1m clustered 3D data and inspected outliers
* used hardware: my laptop with 16 hyperthreaded or 8 physical cores
* used dev env: Visual Studio 2022 (C v14 used) and OpenMP 2.0

Graphical user interface, application, table

Description automatically generated with medium confidenceAll the work related to this project is checked in at [niko-todorov/OMP: My OpenMP experiments. (github.com)](https://github.com/niko-todorov/OMP) along with the previous OpenMP homework experiments. On Windows with VS2022 and OMP 2.0 open the [OMP](https://github.com/niko-todorov/OMP/blob/main/OMP.sln) solution file which will load the following 9 projects, where only the k-means-ser and k-means-omp are applicable to this project:

Figure - OMP solution with the 2 K-Means C projects

# Visualization

Figure - K-Means of 10,000 3D data points, 4 clusters

To visualize the large 3D data sets in 2D I wrote the following 3dscatter.R script:

library(rgl)

data = read.csv("C:/GitHub/OMP/data/cluster\_output\_16\_threads\_dataset.txt", header = FALSE, sep = " ")

# Add a new column with color

mycolors <- c('royalblue', 'darkcyan', 'purple', 'yellow')

data$color <- mycolors[ as.numeric(data$V4+1) ]

# Plot

plot3d(

x=data$`V1`, y=data$`V2`, z=data$`V3`,

col = data$color,type = 's',radius = 3,

xlab="X", ylab="Y", zlab="Z")

# To save to a file:

htmlwidgets::saveWidget(rglwidget(width = 520, height = 520),

file = "./data/3dscatter.html", libdir = "libs", selfcontained = FALSE)

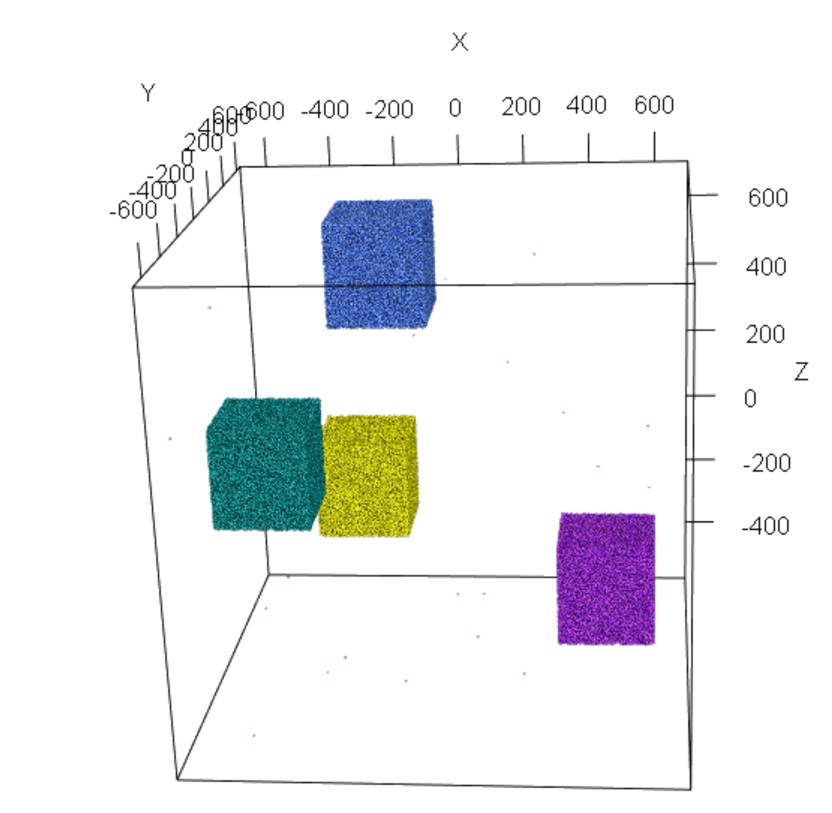


Figure - K-Means of 1,000,000 3D data points, 4 clusters

# Results

## Serial single-threaded run

C:\GitHub\OMP>**k-means-ser**

#Processors available: 8

Enter #Clusters: 4

initial centroid #1: -160.000000,100.000000,410.000000

initial centroid #2: -434.000000,490.000000,-284.000000

initial centroid #3: -305.000000,490.000000,-105.000000

initial centroid #4: -398.000000,585.000000,-202.000000

#1 delta 335040.7734375000

#2 delta 229574.8374023438

#3 delta 32871.3235898076

#4 delta 0.0000163446

#5 delta 0.0000000000

Number of iterations: 5

centroid #1: -184.117294,155.846161,534.946350

centroid #2: -519.011047,-6.172567,21.879932

centroid #3: 459.985443,142.114487,-370.830933

centroid #4: -307.113159,512.938110,-201.735687

Time Taken: **1.184410**

## Parallel 4-threaded run

C:\GitHub\OMP>**k-means-omp**

#Processors available: 8

Enter #Threads: 4

Enter #Clusters: 4

initial centroid #1: -160.000000,100.000000,410.000000

initial centroid #2: -434.000000,490.000000,-284.000000

initial centroid #3: -305.000000,490.000000,-105.000000

initial centroid #4: -398.000000,585.000000,-202.000000

Thread ID:0, start:0, end:250000

Thread ID:1, start:250000, end:500000

Thread ID:2, start:500000, end:750000

Thread ID:3, start:750000, end:1000000

#1 delta 335038.5019531250

#2 delta 229576.0249023438

#3 delta 32869.9293637334

#4 delta 0.0000163486

#5 delta 0.0000000040

Number of iterations :5

centroid #1: -184.117355,155.845108,534.951477

centroid #2: -519.015442,-6.172567,21.879932

centroid #3: 459.982971,142.114731,-370.826843

centroid #4: -307.107697,512.927612,-201.734848

Time Taken: **0.429633**

## Parallel 8-threaded run

C:\GitHub\OMP>**k-means-omp**

#Processors available: 8

Enter #Threads: 8

Enter #Clusters: 4

initial centroid #1: -160.000000,100.000000,410.000000

initial centroid #2: -434.000000,490.000000,-284.000000

initial centroid #3: -305.000000,490.000000,-105.000000

initial centroid #4: -398.000000,585.000000,-202.000000

Thread ID:0, start:0, end:125000

Thread ID:1, start:125000, end:250000

Thread ID:5, start:625000, end:750000

Thread ID:3, start:375000, end:500000

Thread ID:6, start:750000, end:875000

Thread ID:4, start:500000, end:625000

Thread ID:7, start:875000, end:1000000

Thread ID:2, start:250000, end:375000

#1 delta 335037.7421875000

#2 delta 229576.1533203125

#3 delta 32869.4812741608

#4 delta 0.0000161735

#5 delta 0.0000000049

Number of iterations :5

centroid #1: -184.117355,155.845108,534.950806

centroid #2: -519.012817,-6.172567,21.879932

centroid #3: 459.982758,142.114716,-370.827393

centroid #4: -307.107666,512.928711,-201.734833

Time Taken: **0.290386**

## Parallel 16-threaded run

C:\GitHub\OMP>**k-means-omp**

#Processors available: 8

Enter #Threads: 16

Enter #Clusters: 4

initial centroid #1: -160.000000,100.000000,410.000000

initial centroid #2: -434.000000,490.000000,-284.000000

initial centroid #3: -305.000000,490.000000,-105.000000

initial centroid #4: -398.000000,585.000000,-202.000000

Thread ID:0, start:0, end:62500

Thread ID:11, start:687500, end:750000

Thread ID:4, start:250000, end:312500

Thread ID:1, start:62500, end:125000

Thread ID:2, start:125000, end:187500

Thread ID:6, start:375000, end:437500

Thread ID:7, start:437500, end:500000

Thread ID:10, start:625000, end:687500

Thread ID:3, start:187500, end:250000

Thread ID:8, start:500000, end:562500

Thread ID:9, start:562500, end:625000

Thread ID:5, start:312500, end:375000

Thread ID:13, start:812500, end:875000

Thread ID:14, start:875000, end:937500

Thread ID:12, start:750000, end:812500

Thread ID:15, start:937500, end:1000000

#1 delta 335037.7587890625

#2 delta 229576.2348632812

#3 delta 32869.4433096144

#4 delta 0.0000162631

#5 delta 0.0000000161

Number of iterations :5

centroid #1: -184.117371,155.845139,534.950806

centroid #2: -519.012817,-6.172567,21.879932

centroid #3: 459.982788,142.114716,-370.827362

centroid #4: -307.107666,512.928772,-201.734833

Time Taken: **0.393835**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| type | time | iterations | centroid #1 | centroid #2 | centroid #3 | centroid #4 |
| 1-thread | 1.18441 | 5 | -184.117294,155.846161,534.946350 | -519.011047,-6.172567,21.879932 | 459.985443,142.114487,-370.830933 | -307.113159,512.938110,-201.735687 |
| 4-threads | 0.429633 | 5 | -184.117355,155.845108,534.951477 | -519.015442,-6.172567,21.879932 | 459.982971,142.114731,-370.826843 | -307.107697,512.927612,-201.734848 |
| **8-threads** | **0.290386** | **5** | **-184.117355,155.845108,534.950806** | **-519.012817,-6.172567,21.879932** | **459.982758,142.114716,-370.827393** | **-307.107666,512.928711,-201.734833** |
| 16-threads | 0.393835 | 5 | -184.117371,155.845139,534.950806 | -519.012817,-6.172567,21.879932 | 459.982788,142.114716,-370.827362 | -307.107666,512.928772,-201.734833 |

# Conclusion

In conclusion, for the architecture of my Dell XPS 13 laptop, the most optimal K-Means clustering of 1 million 3D points, and K=4 is on 8 threads, followed by 16 and 4 threads (because the 16 hyperthreaded CPUs are in reality 8 physical CPUs). The single-threaded K-Means performed 3-4 times slower than the OpenMP multi-threaded ones.

The 4 centroid coordinates are either identical or nearly identical, which would mean that the points across the different methods got clustered together in identical or near-identical 4 groups. Diff-ing the output cluster indexes of the 1m 3D points there was no difference so I am inclined to attribute the ever-so-slight centroid coordinate differences to truncation/round-off operations.