



**INDIAN INSTITUTE OF INFORMATION TECHNOLOGY
DESIGN AND MANUFACTURING KANCHEEPURAM**

**PROJECT REPORT
ON
K-MEAN CLUSTERING ALGORITHM
USING OPENMP**

**SUBMITTED BY
AMAR KUMAR
(CED17I029)
TO
DR. NOOR MAHAMMAD**

My Laptop Specifications : -

Cpu processor - Intel core i5-7200 CPU @ 2.50GHz

L1d cache:	64 KiB
L1i cache:	64 KiB
L2 cache:	512 KiB
L3 cache:	3 MiB

Thread(s) per core: 2

Core(s) per socket: 2

Socket(s): 1

So number of physical cores = $2 * 1 = 2$

And number of logical cores = $2 * 2 * 1 = 4$

So Total GFLOPS = $1(\text{socket}) * 2(\text{cores}) * 2,500,000,000(\text{cycles per second}) * 4(\text{double precision FLOPs per second}) = 20$
GFLOPS

Problem Statement

A Pizza company wants to open its delivery centres across a very big city.

The challenges they will face is that they need to analyze areas from where pizza is being ordered frequently.

They need to figure out the locations for the pizza stores within all these areas in order to keep the distance between the store and delivery points minimum.

Resolving these challenges includes a lot of analysis and mathematics. We would learn here about how clustering can provide a meaningful and easy method of sorting out such real life challenges.

My Solution to this problem

I have used an unsupervised machine learning algorithm to solve this problem. I have used the K-Means clustering algorithm for solving the above problem.

My dataset contains x and y coordinates of locations of different people/their houses across the city and some randomly generated pizza centers(According to my algorithm, the number of pizza centers can vary from 1 to 10).

We have to find the optimum location for these pizza centers so that each house can get the delivery fastest.

Step1 : For this, I have calculated the distance of every house from every pizza center. The house which is nearest to a particular pizza center, will be served by that particular pizza center in the first iteration.

Step2 : Now the location of the pizza center will change according to the location of houses which come under it. X coordinate of the pizza center will be calculated by taking the mean of x coordinates of the location of houses. Similarly Y coordinate of the pizza center will be calculated by taking the mean of y coordinates of the location of the houses.

Step3 : For further iteration repeat from Step1

Note : More the number of iterations, more optimal will be the location of the pizza center. But the location of the pizza center will not vary after a certain number of iterations as it will start converging to an optimal location of the pizza center such that a house can get its delivery faster.

How did i do parallel programming

I have used openMP technique to parallelize my program.

From my solution to the problem statement, we can clearly see that we have to calculate the distance between pizza center and location of houses. So I have parallelized the program for calculating the distance. My program also contains shifting of pizza centers by taking the mean of x and y coordinates of location of the houses. So I calculated the mean parallelly using the reduction method.

Data Access Optimization

Memory bandwidth(b_{max}) = 7.2 GBytes/sec , peak performance(P_{max}) = 20 GFLOPS

$$\text{Machine Balance}(B_m) = \frac{\text{Memory bandwidth}[Gwords/sec]}{\text{peak performance}[Gflops/sec]} = \frac{7.2}{20} = 0.36 \text{ W/F}$$

In my code floating point operation is being performed while calculating distance which consist of addition, subtraction, multiplication and square root. So there are 4 floating point operations and these floating point operations are being performed on 2 points both having x and y coordinates. So data traffic is 4.

$$\text{Code Balance}(B_c) = \frac{\text{Data Traffic}[words]}{\text{Floating point ops}[Flops]} = \frac{4}{4} = 1$$

$$\text{Computational Intensity} = \frac{1}{\text{Code Balance}} = 1$$

$$\text{Expected maximum fraction of peak performance}(I) = \min\left(1, \frac{B_m}{B_c}\right) = \min(1, 0.36) = 0.36$$

$$\text{Performance} = \text{Expected max fraction of peak performance} * P_{max} = 0.36 * 20 \text{ GFLOPS} = 7.2 \text{ GFlops/sec}$$

Storage Access

I am storing data point i.e location of houses as well as location of pizza centers in 1-Dimensional array and we already know that c like programming language such as c++ implements row major access. So my program is also doing the same i.e. row major access

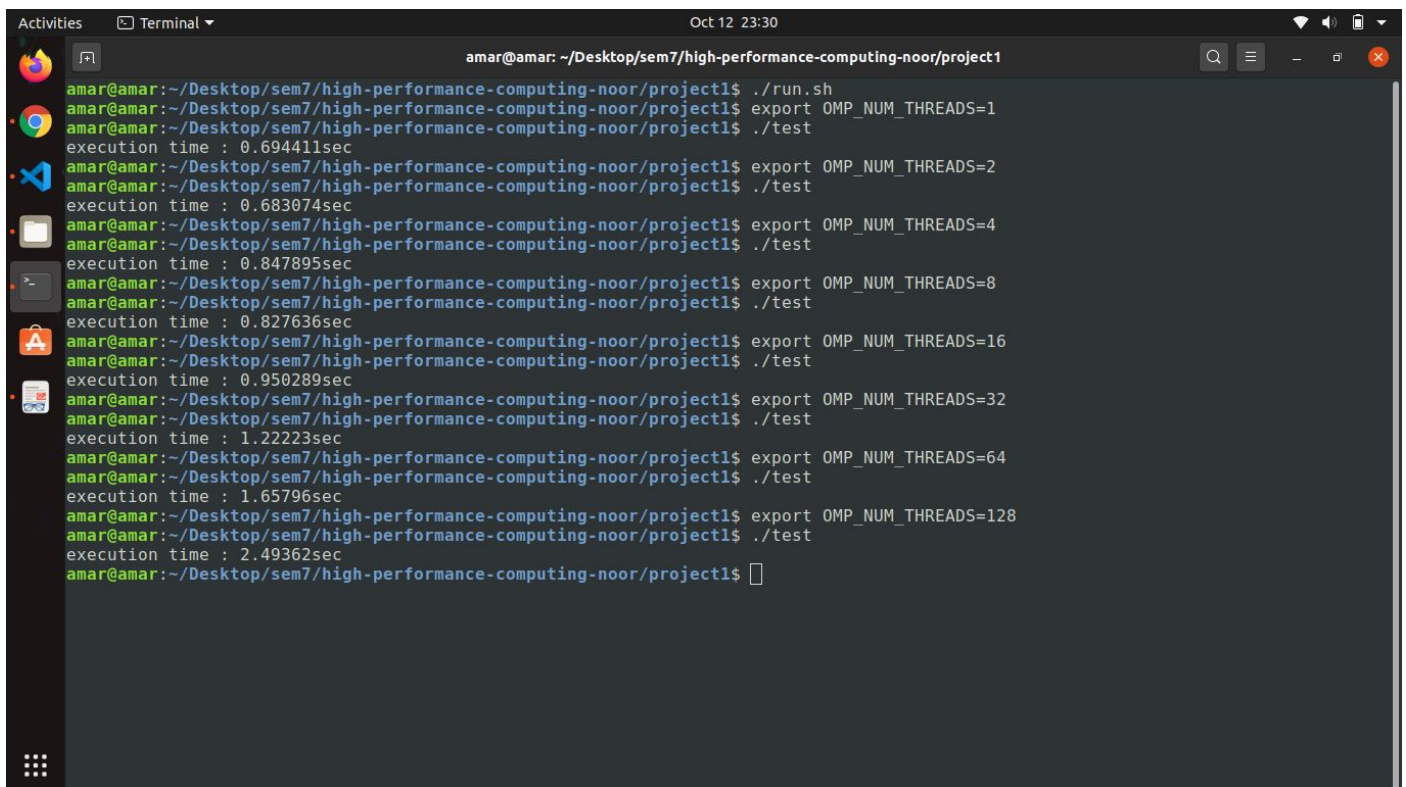
Algorithm classification and access optimization

The optimization potential of many loops on cache-based processors can easily be estimated just by looking at basic parameters like the scaling behavior of data transfers and arithmetic operations versus problem size.

In my code, i am having 4 floating point operations and 4 load/store operation in each iteration as mentioned above and these are also proportional to the my problem size(loop length) N. so if we increase the problem size by two fold then total number of floating point operations and load/store operation will also increase by two fold. Hence my algorithm is classified as $O(N)/O(N)$.

Graph and table

https://docs.google.com/spreadsheets/d/1SdhTE8RQaIKwLZNNdoTpS3lV69wTxATQ51J-Nb_lhQ4/edit?usp=sharing



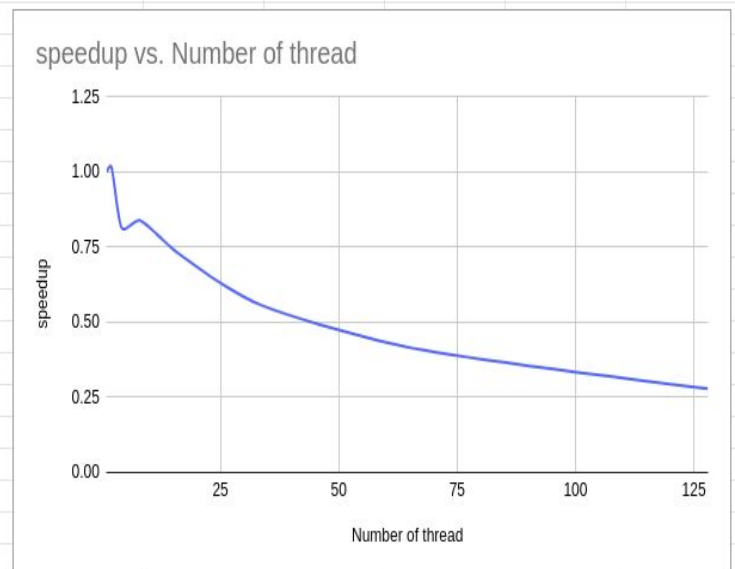
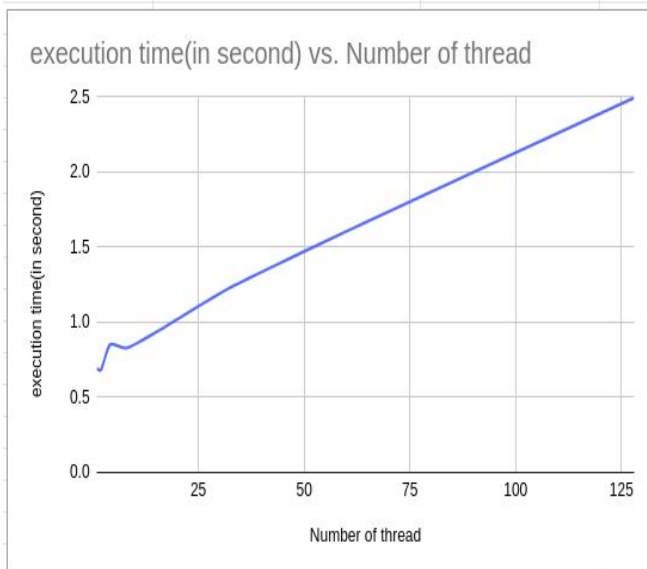
A terminal window screenshot showing the execution of a program with varying thread counts. The user is in the directory `~/Desktop/sem7/high-performance-computing-noor/project1`. The program is run using `./run.sh`, and the execution time is measured for different values of `OMP_NUM_THREADS` (1, 2, 4, 8, 16, 32, 64, 128). The execution time increases as the number of threads increases, but the rate of increase slows down as the thread count goes higher, indicating some level of parallelization.

```
amar@amar: ~/Desktop/sem7/high-performance-computing-noor/project1
amar@amar:~/Desktop/sem7/high-performance-computing-noor/project1$ ./run.sh
amar@amar:~/Desktop/sem7/high-performance-computing-noor/project1$ export OMP_NUM_THREADS=1
amar@amar:~/Desktop/sem7/high-performance-computing-noor/project1$ ./test
execution time : 0.694411sec
amar@amar:~/Desktop/sem7/high-performance-computing-noor/project1$ export OMP_NUM_THREADS=2
amar@amar:~/Desktop/sem7/high-performance-computing-noor/project1$ ./test
execution time : 0.683074sec
amar@amar:~/Desktop/sem7/high-performance-computing-noor/project1$ export OMP_NUM_THREADS=4
amar@amar:~/Desktop/sem7/high-performance-computing-noor/project1$ ./test
execution time : 0.847895sec
amar@amar:~/Desktop/sem7/high-performance-computing-noor/project1$ export OMP_NUM_THREADS=8
amar@amar:~/Desktop/sem7/high-performance-computing-noor/project1$ ./test
execution time : 0.827636sec
amar@amar:~/Desktop/sem7/high-performance-computing-noor/project1$ export OMP_NUM_THREADS=16
amar@amar:~/Desktop/sem7/high-performance-computing-noor/project1$ ./test
execution time : 0.950289sec
amar@amar:~/Desktop/sem7/high-performance-computing-noor/project1$ export OMP_NUM_THREADS=32
amar@amar:~/Desktop/sem7/high-performance-computing-noor/project1$ ./test
execution time : 1.22223sec
amar@amar:~/Desktop/sem7/high-performance-computing-noor/project1$ export OMP_NUM_THREADS=64
amar@amar:~/Desktop/sem7/high-performance-computing-noor/project1$ ./test
execution time : 1.65796sec
amar@amar:~/Desktop/sem7/high-performance-computing-noor/project1$ export OMP_NUM_THREADS=128
amar@amar:~/Desktop/sem7/high-performance-computing-noor/project1$ ./test
execution time : 2.49362sec
amar@amar:~/Desktop/sem7/high-performance-computing-noor/project1$
```

K mean clustering algorithm

number of data points = 1000 , number of iterations = 1000

Number of thread	execution time(in second)	speedup	parallelization fraction(f)
1	0.694411	1	0
2	0.683074	1.01659703	0.03265213253
4	0.847895	0.8189823032	-0.2947034729
8	0.827636	0.8390294767	-0.2192608453
16	0.950289	0.7307366496	-0.3930475372
32	1.22223	0.568150839	-0.7846151909
64	1.65796	0.4188345919	-1.409602424
128	2.49362	0.2784750684	-2.611387204



Calculation of parallelization fraction

$T(1) = 0.694411$ seconds

Here , for $P = 2$ the execution time is minimum

$T(P) = 0.683074$ seconds

$$\text{Speedup} = \frac{T(1)}{T(P)} = \frac{130564}{105536} = 1.237151304.$$

From Amdahl's Law,

Speedup = $\frac{1}{(f/P) + (1-f)}$ Where , f = Parallelization factor P = Thread Number

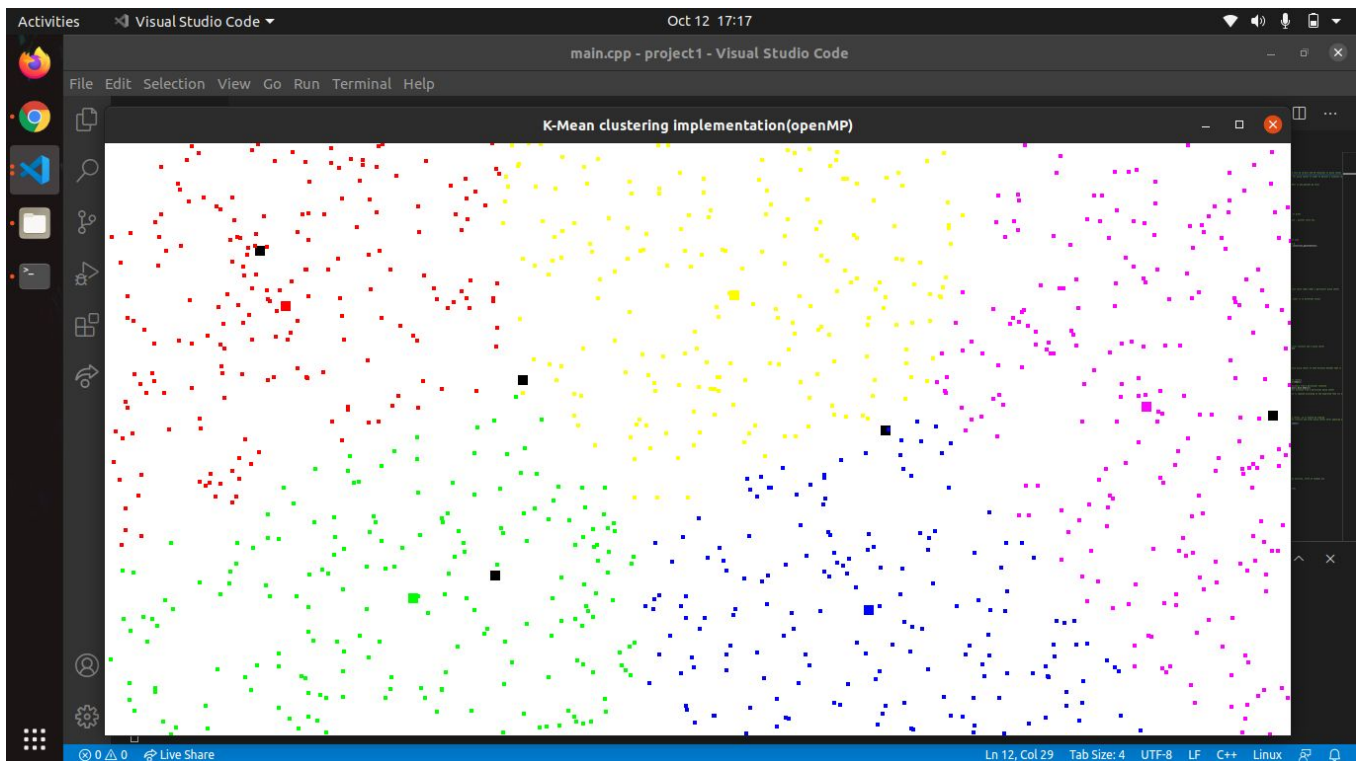
$$\text{So, } f = \frac{(1-T(P)/T(1))}{(1-(1/P))}$$

Therefore, f = 0.03265213253 which means that approx. 3.26% of the program is parallelizable.

Addition things done by me

It is somewhat difficult to realize the shifting of pizza centers as well as the location of houses which comes under its service. So in order to visualize these things I have used a library called GLUT in c++. This library helps us to plot the points on a screen so that we can visualize the location of houses which come under service of a particular pizza center. I have also coloured the locations of houses and its nearest pizza center in a unique colour and i have kept the pizza center point size more than the location of houses in order to differentiate between pizza center and location of houses.

My optimum placement of pizza centers and area under its service is shown below for 5 pizza centers and 1000 location of houses for 15 iterations

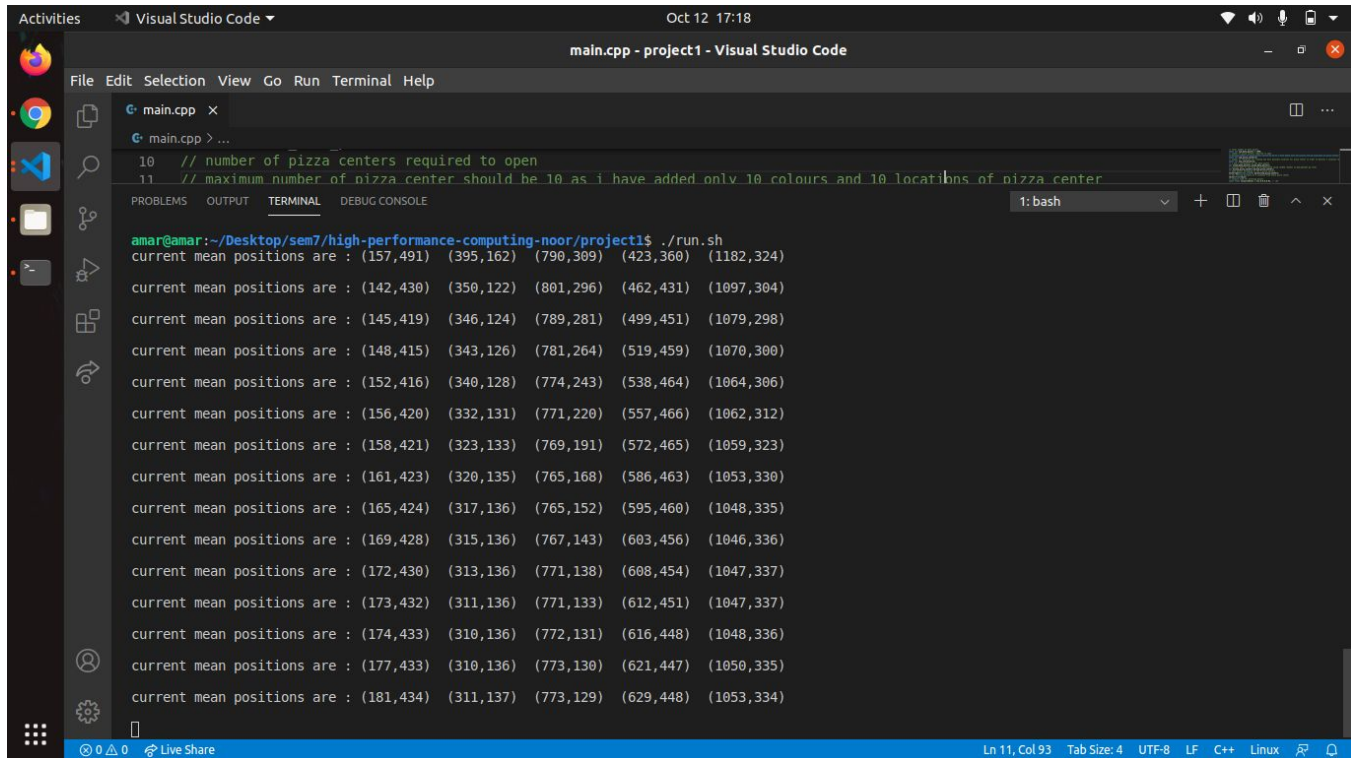


Bold black-colour-points are the randomly generated initial location of pizza centers which got changed in 15 iterations.

Bold coloured-points are optimal location of pizza centers

All the small coloured-points represent the location of houses

The change of location of 5 pizza centers in 15 iterations are as follows :-



```
main.cpp - project1 - Visual Studio Code
10 // number of pizza centers required to open
11 // maximum number of pizza center should be 10 as i have added only 10 colours and 10 locations of pizza center

amar@amar:~/Desktop/sem7/high-performance-computing-noor/project1$ ./run.sh
current mean positions are : (157,491) (395,162) (790,309) (423,360) (1182,324)
current mean positions are : (142,430) (350,122) (801,296) (462,431) (1097,304)
current mean positions are : (145,419) (346,124) (789,281) (499,451) (1079,298)
current mean positions are : (148,415) (343,126) (781,264) (519,459) (1070,300)
current mean positions are : (152,416) (340,128) (774,243) (538,464) (1064,306)
current mean positions are : (156,420) (332,131) (771,220) (557,466) (1062,312)
current mean positions are : (158,421) (323,133) (769,191) (572,465) (1059,323)
current mean positions are : (161,423) (320,135) (765,168) (586,463) (1053,330)
current mean positions are : (165,424) (317,136) (765,152) (595,460) (1048,335)
current mean positions are : (169,428) (315,136) (767,143) (603,456) (1046,336)
current mean positions are : (172,430) (313,136) (771,138) (608,454) (1047,337)
current mean positions are : (173,432) (311,136) (771,133) (612,451) (1047,337)
current mean positions are : (174,433) (310,136) (772,131) (616,448) (1048,336)
current mean positions are : (177,433) (310,136) (773,130) (621,447) (1050,335)
current mean positions are : (181,434) (311,137) (773,129) (629,448) (1053,334)
```