National University
of computer and emerging sciences

Operating Systems Project Report

Group Members

Aheed Tahir 21K-4517
Abdul Haseeb 21K-3217
Ashad Abdullah 21K-3296

Teacher

Dr. Nadeem Kafi

K-Means Clustering:

Parallelism vs Sequential

using **OpenMp and Pthreads**

17th March - 7th May 2023

Abstract (Introduction and Background)

Our project implements the K-Means clustering algorithm using both sequential and parallel (OpenMP and Pthreads) programming techniques in C++ and compares the efficiency of both approaches. K-Means is an unsupervised clustering algorithm that segments data based on the similarity between data points.

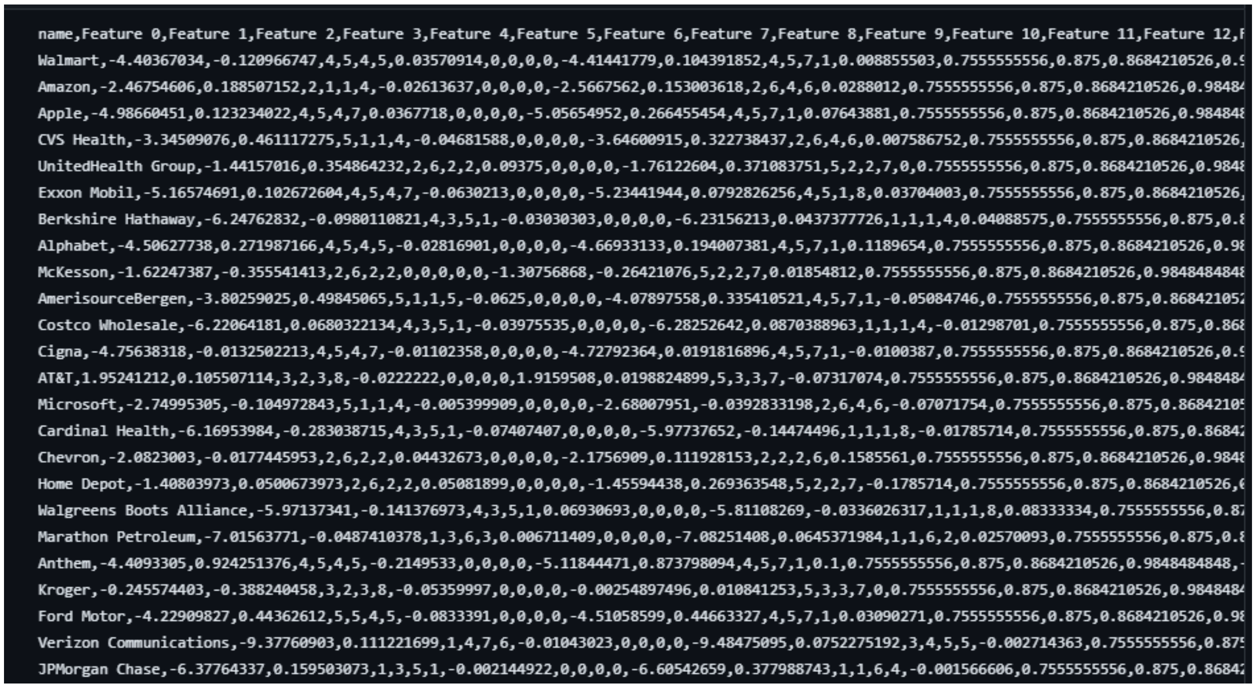
The parallel programming technique we used is OpenMP and Pthreads, which allows us to efficiently utilize multiple threads to speed up the execution of the algorithm.

Implementation Of K-Means Clustering algorithm

We have shown the implementation through simulating and visualizing the working of a stock market's opening and closings. We first scraped a day's worth of stock data from a Fortune 500 website, normalized it, and then applied the K-Means algorithm. Ashad implemented the K-Means Clustering algorithm sequentially. The algorithm is used to group the stocks based on their similarities, which are measured using Euclidean distance.

The data set used in our model includes a total of **385** different parameters, which are used to predict stock market trends and inform investment decisions. These parameters include key features such as, opening and closing prices, volume, and income, PB Ratio as well as a variety of other metrics such as market capitalization, price-to-earnings ratio, and historical performance data, among others

Providing that, the data set of our model looked like this (to view full data visit this:



```
name,Feature 0,Feature 1,Feature 2,Feature 3,Feature 4,Feature 5,Feature 6,Feature 7,Feature 8,Feature 9,Feature 10,Feature 11,Feature 12,
Walmart,-4.40367034,-0.120966747,4,5,4,5,0.03570914,0,0,0,0,-4.41441779,0.104391852,4,5,7,1,0.008855503,0.7555555556,0.875,0.8684210526,0.
Amazon,-2.46754606,0.188507152,2,1,1,4,-0.02613637,0,0,0,0,-2.5667562,0.153003618,2,6,4,6,0.0288012,0.7555555556,0.875,0.8684210526,0.9848
Apple,-4.98660451,0.123234022,4,5,4,7,0.0367718,0,0,0,0,-5.05654952,0.266455454,4,5,7,1,0.07643881,0.7555555556,0.875,0.8684210526,0.98484
CVS Health,-3.34509076,0.461117275,5,1,1,4,-0.04681588,0,0,0,0,-3.64600915,0.322738437,2,6,4,6,0.007586752,0.7555555556,0.875,0.8684210526,
UnitedHealth Group,-1.44157016,0.354864232,2,6,2,2,0.09375,0,0,0,0,-1.76122604,0.371083751,5,2,2,7,0,0.7555555556,0.875,0.8684210526,0.984
Exxon Mobil,-5.16574691,0.102672604,4,5,4,7,-0.0630213,0,0,0,0,-5.23441944,0.0792826256,4,5,1,8,0.03704003,0.7555555556,0.875,0.8684210526,
Berkshire Hathaway,-6.24762832,-0.0980110821,4,3,5,1,-0.03030303,0,0,0,0,-6.23156213,0.0437377726,1,1,1,4,0.04088575,0.7555555556,0.875,0.
Alphabet,-4.50627738,0.271987166,4,5,4,5,-0.02816901,0,0,0,0,-4.66933133,0.194007381,4,5,7,1,0.1189654,0.7555555556,0.875,0.8684210526,0.9
McKesson,-1.62247387,-0.355541413,2,6,2,2,0,0,0,0,0,-1.30756868,-0.26421076,5,2,2,7,0.01854812,0.7555555556,0.875,0.8684210526,0.984848484
AmerisourceBergen,-3.80259025,0.49845065,5,1,1,5,-0.0625,0,0,0,0,-4.07897558,0.335410521,4,5,7,1,-0.05084746,0.7555555556,0.875,0.86842105
Costco Wholesale,-6.22064181,0.0680322134,4,3,5,1,-0.03975535,0,0,0,0,-6.28252642,0.0870388963,1,1,1,4,-0.01298701,0.7555555556,0.875,0.86
Cigna,-4.75638318,-0.0132502213,4,5,4,7,-0.01102358,0,0,0,0,-4.72792364,0.0191816896,4,5,7,1,-0.0100387,0.7555555556,0.875,0.8684210526,0.
AT&T,1.95241212,0.105507114,3,2,3,8,-0.0222222,0,0,0,0,1.9159508,0.0198824899,5,3,3,7,-0.07317074,0.7555555556,0.875,0.8684210526,0.984848
Microsoft,-2.74995305,-0.104972843,5,1,1,4,-0.005399909,0,0,0,0,-2.68007951,-0.0392833198,2,6,4,6,-0.07071754,0.7555555556,0.875,0.868421
Cardinal Health,-6.16953984,-0.283038715,4,3,5,1,-0.07407407,0,0,0,0,-5.97737652,-0.14474496,1,1,1,8,-0.01785714,0.7555555556,0.875,0.8684
Chevron,-2.0823003,-0.0177445953,2,6,2,2,0.04432673,0,0,0,0,-2.1756909,0.111928153,2,2,2,6,0.1585561,0.7555555556,0.875,0.8684210526,0.984
Home Depot,-1.40803973,0.0500673973,2,6,2,2,0.05081899,0,0,0,0,-1.45594438,0.269363548,5,2,2,7,-0.1785714,0.7555555556,0.875,0.8684210526,
Walgreens Boots Alliance,-5.97137341,-0.141376973,4,3,5,1,0.06930693,0,0,0,0,-5.81108269,-0.0336026317,1,1,1,8,0.08333334,0.7555555556,0.8
Marathon Petroleum,-7.01563771,-0.0487410378,1,3,6,3,0.006711409,0,0,0,0,-7.08251408,0.0645371984,1,1,6,2,0.02570093,0.7555555556,0.875,0.
Anthem,-4.4093305,0.924251376,4,5,4,5,-0.2149533,0,0,0,0,-5.11844471,0.873798094,4,5,7,1,0.1,0.7555555556,0.875,0.8684210526,0.9848484848,
Kroger,-0.245574403,-0.388240458,3,2,3,8,-0.05359997,0,0,0,0,-0.00254897496,0.010841253,5,3,3,7,0,0.7555555556,0.875,0.8684210526,0.984848
Ford Motor,-4.22909827,0.44362612,5,5,4,5,-0.0833391,0,0,0,0,-4.51058599,0.44663327,4,5,7,1,0.03090271,0.7555555556,0.875,0.8684210526,0.9
Verizon Communications,-9.37760903,0.111221699,1,4,7,6,-0.01043023,0,0,0,0,-9.48475095,0.0752275192,3,4,5,5,-0.002714363,0.7555555556,0.87
JPMorgan Chase,-6.37764337,0.159503073,1,3,5,1,-0.002144922,0,0,0,0,-6.60542659,0.377988743,1,1,6,4,-0.001566606,0.7555555556,0.875,0.8684
```

Inputs

The data is read from a file using python and normalized initially.

Regular Inputs include reading from the updated file using c++ giving inputs K which are the number of clusters and N which are the number of stocks you want to read from the dataset.

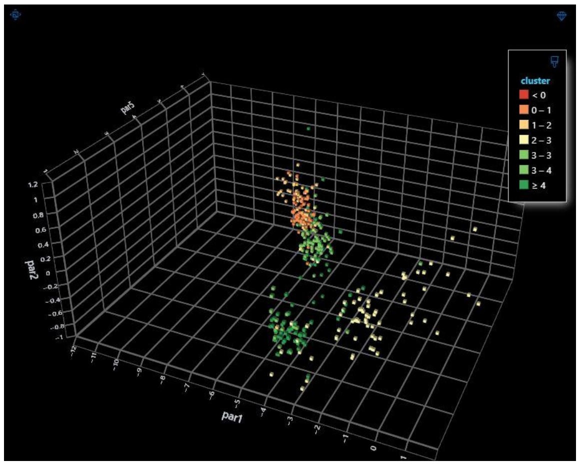
Output

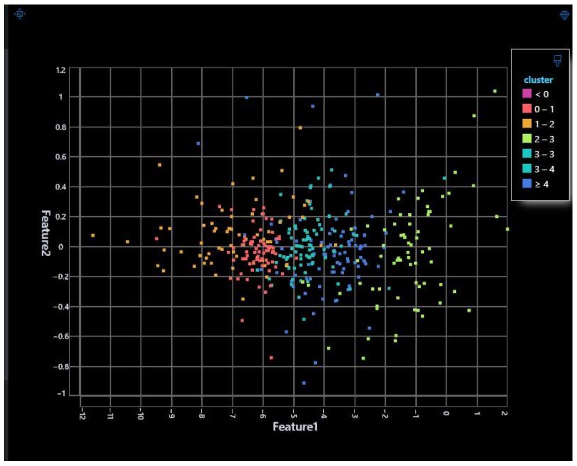
Our Code Outputs the Clusters and Data Points created which are different for each type of code.

This data is used to visualize the graphs.

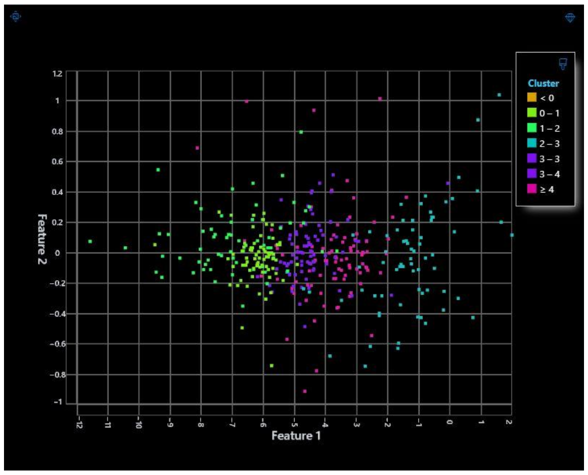
The Outputs Can Be Visualized Into Scatter Plots As Shown

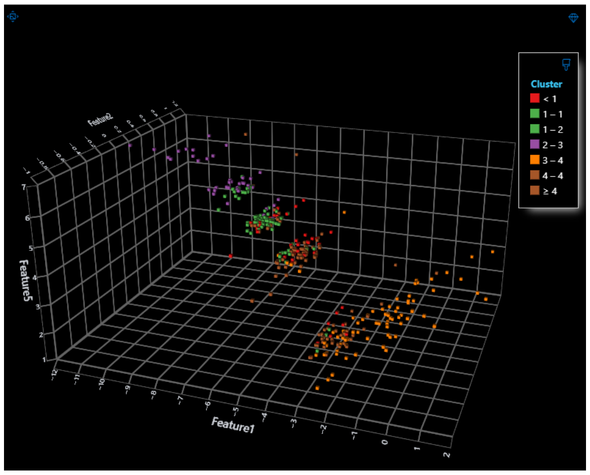
Sequential:



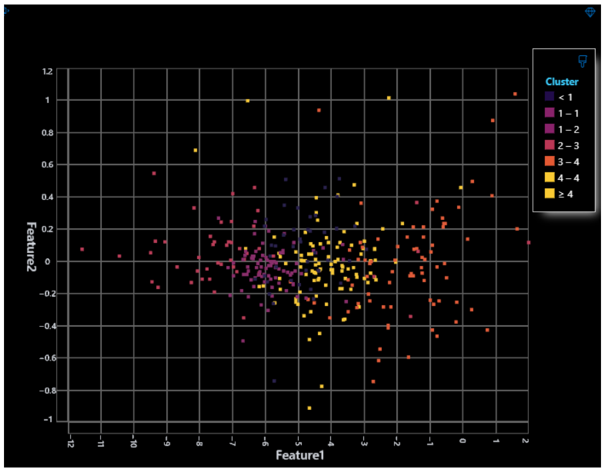


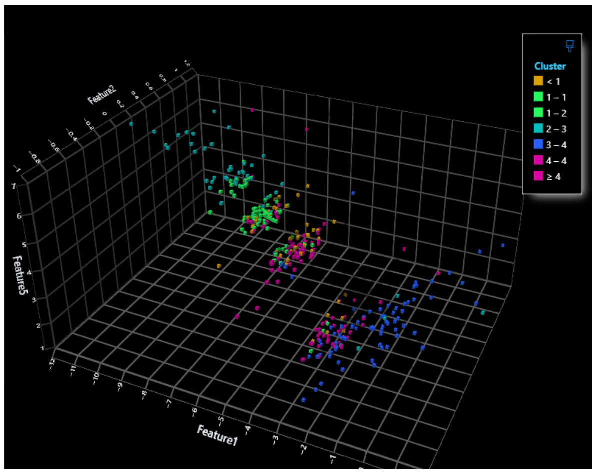
OpenMp:



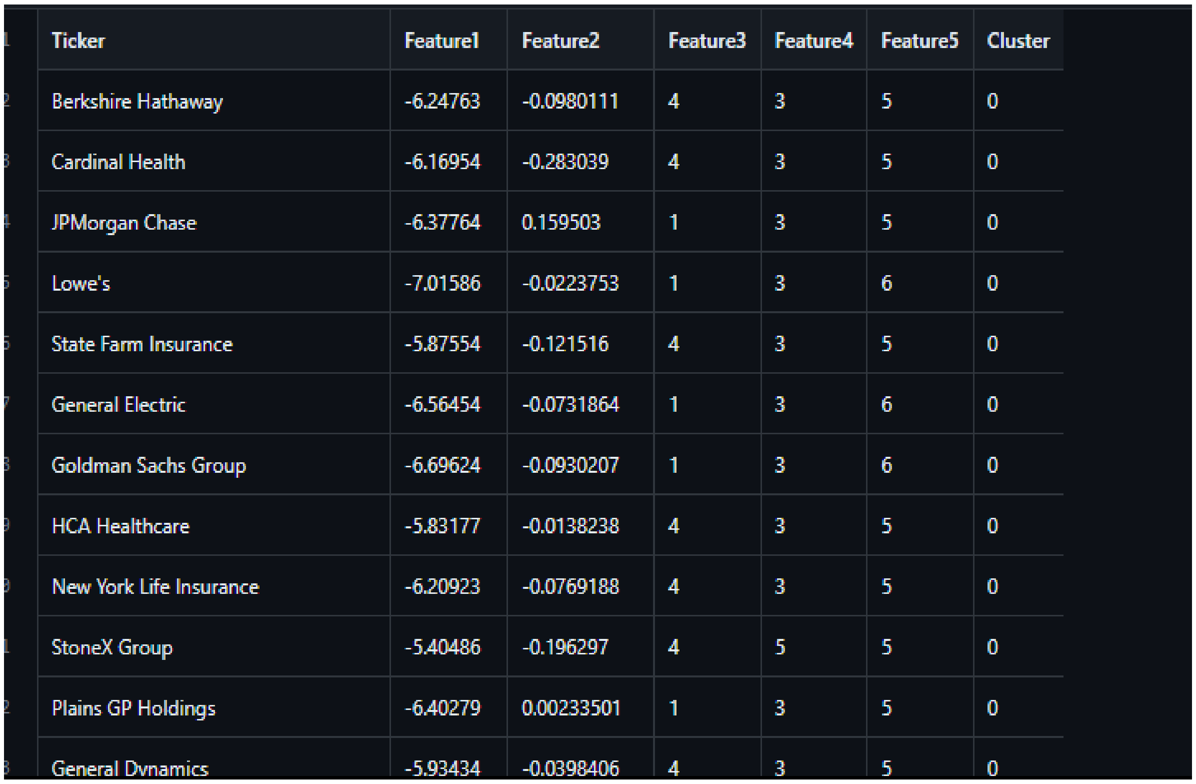


Pthread:





The sample clusters are given in this output file



Ticker	Feature1	Feature2	Feature3	Feature4	Feature5	Cluster
Berkshire Hathaway	-6.24763	-0.0980111	4	3	5	0
Cardinal Health	-6.16954	-0.283039	4	3	5	0
JPMorgan Chase	-6.37764	0.159503	1	3	5	0
Lowe's	-7.01586	-0.0223753	1	3	6	0
State Farm Insurance	-5.87554	-0.121516	4	3	5	0
General Electric	-6.56454	-0.0731864	1	3	6	0
Goldman Sachs Group	-6.69624	-0.0930207	1	3	6	0
HCA Healthcare	-5.83177	-0.0138238	4	3	5	0
New York Life Insurance	-6.20923	-0.0769188	4	3	5	0
StoneX Group	-5.40486	-0.196297	4	5	5	0
Plains GP Holdings	-6.40279	0.00233501	1	3	5	0
General Dynamics	-5.93434	-0.0398406	4	3	5	0

We couldn't have shown all 385 parameters

Architecture and Design

Each code has the following attributes and methods:

The First Two Functions Played the Main Role for Both Sequential and Parallel

1. *Meanrecompute(): Recompute the centroids for each cluster by taking the mean of all points in the cluster*
2. *AssignClusters(): Assign each point to the closest cluster*
3. *divideTwo(): Divide the sum of the stock parameters by the count to calculate the average.*
4. *addTwo(): Add the parameters of two stocks.*
5. *computeDistance(): Calculate the Euclidean distance between two stock instances.*

For Openmp these two functions were changed:

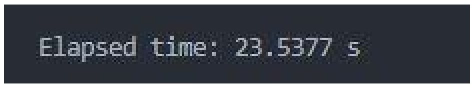
1. *AssignClusters(): was wrapped in* `#pragma omp parallel`
2. *The For Loops for this function were also parallelized using* `#pragma omp for`
3. *MeanRecompute(): also had* `#pragma omp parallel for` *to parallelize it*

Results and Discussion

We have used sanddance to visualize the three different outputs expected from the different codes, and generated scatter plots in 3D to help easily understand the efficiency of the codes.

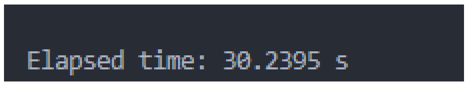
Furthermore below attached are possible outputs of our codes:

Time Elapsed For Sequential Code Run:



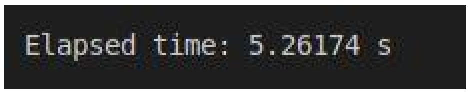
```
Elapsed time: 23.5377 s
```

Time Elapsed For Pthread Code Run:



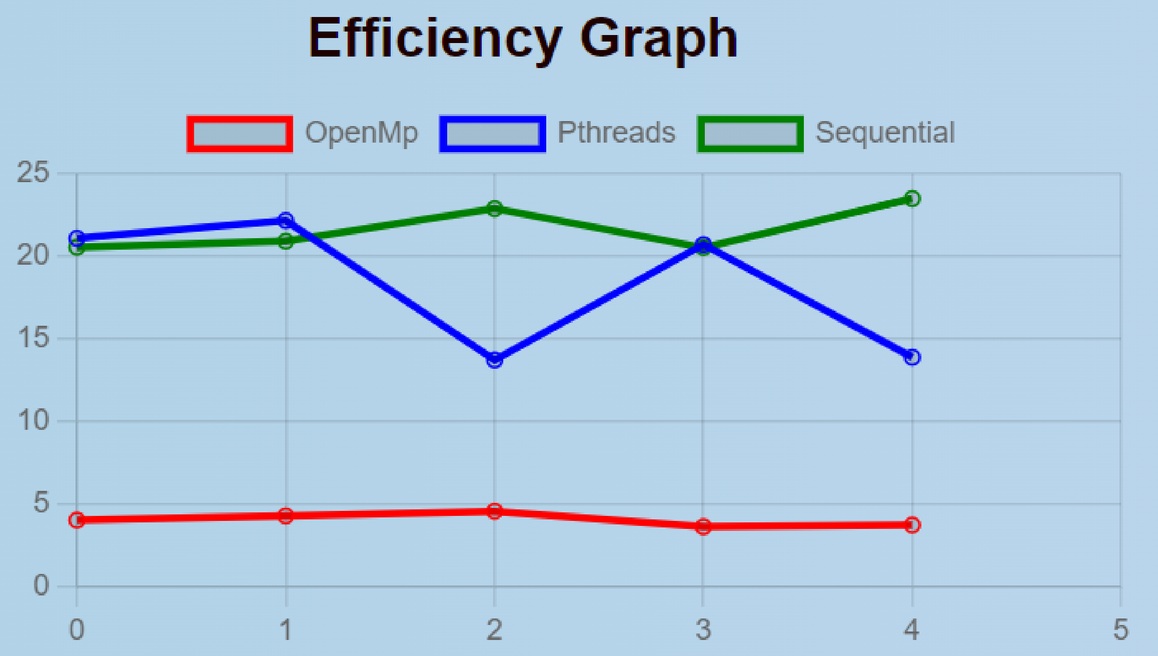
```
Elapsed time: 30.2395 s
```

Time Elapsed For OpenMp Code Run:



```
Elapsed time: 5.26174 s
```

Averaging Out the time taken for each code after multiple tries we were able to come up with a graph to show the comparison visually:



This graph shows the Efficiency of different methods to attempt the K-Means Clustering Algorithm.

The OpenMp method is clearly the most efficient in our case with the lowest times as seen , although the data set used for it was significanlty smaller

In conclusion, this project demonstrates the application of the K-Means Clustering algorithm in an operating systems context for stock market analysis. By parallelizing the algorithm, we show that improved performance and scalability can be achieved using OpenMP and Pthreads. Furthermore, the user interface allows for effective comparison and visualization of the different implementations, providing a clear understanding of their performance characteristics.

UI for Code Comparison and Visualization:

Aheed Tahir designed a user interface layout based on Html/Css/Js that allows users to compare the performance of different K-Means implementations and visualize the clustering results. This layout provides an intuitive and informative way to analyze the performance and efficiency of the implemented algorithms. Code Comparison, Visualization(the graph above) and Team Contributions are part of the Ui which can be shown with the project evaluation.

Our Project GitHub URL

K-Means Clustering