APPM4058A & APPM7045A Project: K-Means Clustering & Fuzzy C-Means Clustering

1846346, 1669326, COMS Hons

June 27, 2021

Problem Domain

Fuzzy C-Means Clustering and K-Means clustering are algorithms that are typically used in semi-supervised & unsupervised Machine Learning. K-Means clustering aims to segment N observations into K clusters where each observation belongs to the cluster with the nearest mean. It aims to minimise within cluster variance. K-Means clustering (Dunn [1973]) can be used for cluster analysis or feature learning. The problem is considered NP-Hard. Fuzzy C-Means clustering (Macqueen [1967]) is considered a soft version of K-Means clustering where the observation has associations to multiple clusters rather than just one. Fuzzy C-Means clustering can be used in image analysis and marketing. Given that these algorithms are computationally expensive & computation can be conducted in parallel. This makes the use of parallel programming techniques highly desirable. Thus, we will explore the effects of using CUDA[1] & Open MPI[2] frameworks on computation time.

Definitions The following definitions are relevant in this report:

- Observation: A data point in d-dimensional space that has an associated cluster/s.
- K-Mean: A reference point that is designated to be a d-dimensional cluster. To which observations are assigned. They represent the average location of their assigned clusters.
- C-Mean: A reference point with some degree of fuzziness. It is a d-dimensional cluster. All observations will have some level of association p, where $p \in [0,1]$ to the reference point. They represent the average location based on observations' associations.

Methodology

For each algorithm, we will give a brief overview of the pseudocode. The pseudocode will be implemented in our serial versions. We will highlight the components of the serial code that will be modified to make use of parallel methods for an improved performance.

[1] https://developer.nvidia.com/cuda-gpus
[2] https://www.open-mpi.org/

K-Means Clustering

1: *Given N observations & K Means*
2: *Initialise K Means at random points in d-dimensional space*
3: *For i=0,1,...,iterations*
4: *Calculate distance to each Mean for each observation*
5: *Assign observation to closest Mean*
6: *Update the Means*
7: *Return observations*

Fuzzy C-Means Clustering

1: *Given N observations & C Means*
2: *Initialise C Means at random points in d-dimensional space*
3: *For i=0,1,...,iterations*
4: *Calculate distance to each Mean for each observation*
5: *Update association of each data point to an observation*
6: *Update the C-Means centres*
7: *Return observations*

For K-Means clustering we will parallelise the assignment of observations to each cluster. In CUDA we will aim to assign the maximum number of threads. Using this, each thread will take a single observation and calculate the distance to each cluster. This will update the observations assignment to a cluster. In Open MPI we will aim to apply a similar approach. Here we will distribute the observations to each process such that each process receives $\frac{Number\ of\ Observations}{Number\ of\ Processes}$, we will refer to this number as A. Each process calculates the distances for their A observations and assigns them to the relevant cluster. From here the averaging of the clusters is done in 2 steps. The first is that every node will calculate a partial sum of where the node should be, the master node then takes the sum of the partial sums and averages it. The new clusters are then broadcasted to each node and the process repeats.

For Fuzzy C-Means clustering we parallelise the calculations in a similar fashion to that of K-Means clustering. In CUDA we will aim to assign the maximum number of threads to update our observations association to various clusters. This requires a distance calculation similar to our K-Means methods however, here we need to make use of

the power function with the exponent being the level of fuzziness. We have set this level of fuzziness to be 4. Thus, this is vastly more computationally intensive compared to the K-Means method. To update the C clusters we will aim to spawn as many threads as there are clusters, thus limiting us to 1024 clusters. Each thread will update a given cluster using the observations.

In Open MPI we will spawn 8 processes. Each process will calculate A observations' cluster association. To update our C clusters we will compute partial sums of the *point sum* & *prob sum*, based on the Fuzzy C-Means algorithm. Each process will then accumulate their results for these partial sums into a single process - the rank 0 process. Using this cumulative result of partial sums we will calculate C-Means updated value.

Validation of Results

In order to ensure that our parallel methods produce the same results as our serial methods we will do the following for each algorithm:

- Create a serial implementation.
- Initialise our synthetic data.
- Store two sets of the synthetic data at runtime, one set will be used to perform the parallel computations.
- Maintain the order of parallel computations to be the same as that of the serial computations.
- Collect all parallel results into a single result.
- Compute the serial version using the second set of synthetic data.
- Compare each cluster and verify that they are of the same value, or within an acceptable tolerance level.

Aims

Our aim is to show that a parallel implementation for both the Fuzzy C-Means algorithm & K-Means algorithm provides significant benefits in computational performance. We will showcase the benefits of using CUDA & Open MPI and how these two methods of parallelisation compare to each other.

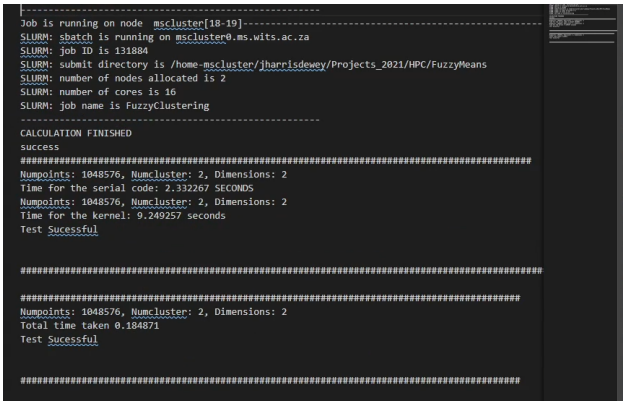


Figure 1: Proof of Cluster Usage

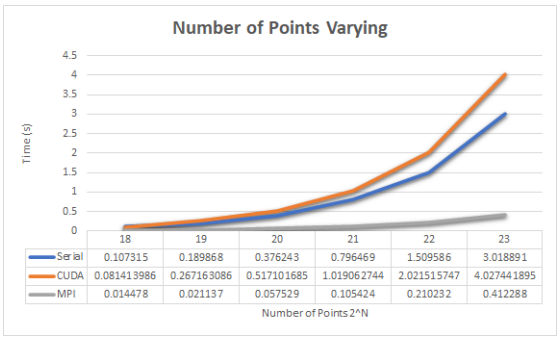
Experimental Setup

We performed our computations with computers that had the following noteworthy hardware: Intel Core i9-10940X CPU (14 cores), NVIDIA RTX3090 GPU (24GB), and 128GB of system RAM. We will aim to utilise the maximum number of threads in CUDA implementations. We will aim to use 8 processors for our Open MPI implementations. Computations were performed using High Performance Computing infrastructure provided by the Mathematical Sciences Support unit at the University of the Witwatersrand.

Evaluation of Results & Discussion

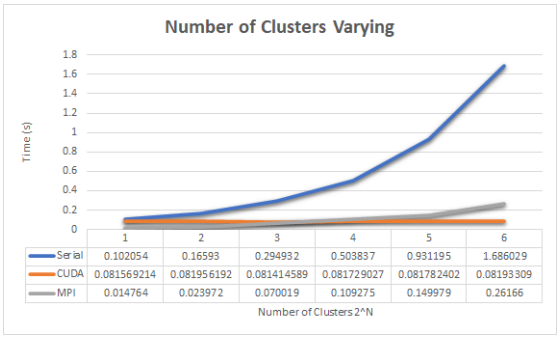
Results

K-Means



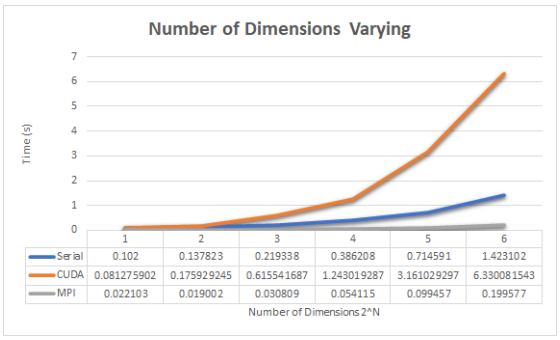
	18	19	20	21	22	23
Serial	0.107315	0.189868	0.376243	0.796469	1.509586	3.018891
CUDA	0.081413986	0.267163086	0.517101685	1.019062744	2.021515747	4.027441895
MPI	0.014478	0.021137	0.057529	0.105424	0.210232	0.412288

Figure 2: K-Means: Number of Points Varying for 2 Dimensions and 2 Clusters



	1	2	3	4	5	6
Serial	0.102054	0.16593	0.294932	0.503837	0.931195	1.686029
CUDA	0.081569214	0.081956192	0.081414589	0.081729027	0.081782402	0.08193309
MPI	0.014764	0.023972	0.070019	0.109275	0.149979	0.26166

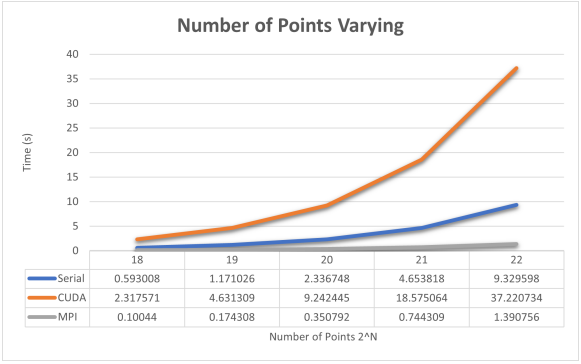
Figure 3: K-Means: Number of Clusters Varying for 2 Dimensions and 2^{18} Points



	1	2	3	4	5	6
Serial	0.102	0.137823	0.219338	0.386208	0.714591	1.423102
CUDA	0.081275902	0.175929245	0.615541687	1.243019287	3.161029297	6.330081543
MPI	0.022103	0.019002	0.030809	0.054115	0.099457	0.199577

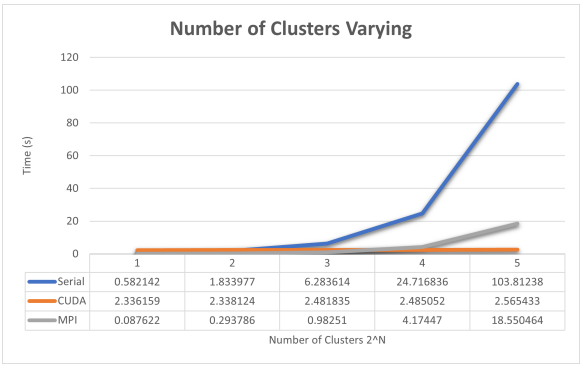
Figure 4: K-Means: Number of Dimensions Varying for 2 Dimensions and 2^{18} Points

Fuzzy C-Means



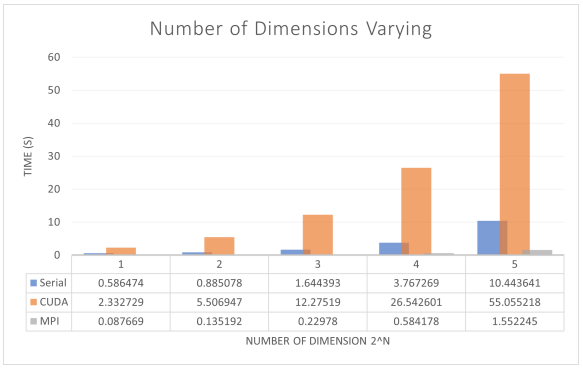
	18	19	20	21	22
Serial	0.593008	1.171026	2.336748	4.653818	9.329598
CUDA	2.317571	4.631309	9.242445	18.575064	37.220734
MPI	0.10044	0.174308	0.350792	0.744309	1.390756

Figure 5: Fuzzy C-Means: Number of Clusters Varying for 2 Dimensions and 2^{18} Points



	1	2	3	4	5
Serial	0.582142	1.833977	6.283614	24.716836	103.81238
CUDA	2.336159	2.338124	2.481835	2.485052	2.565433
MPI	0.087622	0.293786	0.98251	4.17447	18.550464

Figure 6: Fuzzy C-Means: Number of Clusters Varying for 2 Dimensions and 2^{18} Points



	1	2	3	4	5
Serial	0.586474	0.885078	1.644393	3.767269	10.443641
CUDA	2.332729	5.506947	12.27519	26.542601	55.055218
MPI	0.087669	0.135192	0.22978	0.584178	1.552245

Figure 7: Fuzzy C-Means: Number of Dimensions Varying for 2 Dimensions and 2^{18} Points

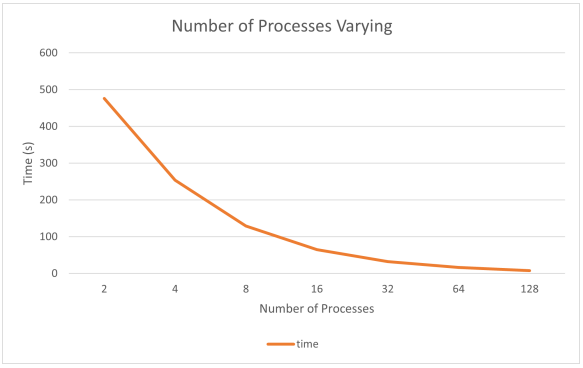


Figure 8: Fuzzy C-Means: Number of Processes Varying for 2^3 Clusters, 2^3 Dimensions and 2^{24} Points

Discussion of Results

K-Means

Varying Points: Based on the above results (Fig.2) for K-Mean clustering, we can see that as the number of points varies, CUDA gives the worst times, whilst Open MPI gives the best time. The CUDA implementation is based on a

global memory approach and therefore there is a lot of contention as each thread is attempting to access the clusters position. We give a discussion and possible solution about this below in the Limitations & Difficulties section. The Open MPI solution is the best in this case and this makes sense. As the number of points changes the entire problem is being split over multiple nodes. This allows for a massive parallelisation speedup.

Varying Clusters: We can see that as we vary the number of clusters Fig.3, the CUDA implementation remains almost constant whilst the serial implementation starts to get exponentially worse. This makes sense since in the CUDA implementation, we assign each cluster to its own thread. This is so that the calculation is happening completely in parallel. This allows us to get a massive parallel advantage compared to the sequential version. The Open MPI version of this also performs well, but is slightly worse than the CUDA version.

Varying Dimensions: Finally, for varying the number of dimensions we can see (Fig.4) that CUDA performs the worst here since there is no explicit code to parallelise the loop over the dimensions. So changing the dimensions of the data causes each thread to stay in a for loop for longer. This causes a larger data contention when each thread is accessing the cluster position as it does this more often. Open MPI gives the best results here, as the number of points is split over multiple nodes. The Open MPI code also has no contention over the accessing of a clusters position and hence performs the best.

Fuzzy C-Means

Varying Points: Based on the above results (Fig.5) for Fuzzy C-Means, we can see that the graphs give a similar output to the K-Means implementation, but just at a longer run time for everything. Fuzzy C-Means is a more expensive calculation than K-Means clustering and hence why the times shown are larger. The overall trend of the graphs remain the same though. The CUDA implementation as shown in the graphs performs the worst, but this is due to the data contention as mentioned before in K-Means. We give a discussion and possible solution about this in the Limitations & Difficulties section below. The Open MPI solution performed the best as the problem was distributed over multiple nodes without the data contention issue.

Varying Clusters: We can see that as we vary the number of clusters (Fig.6), the CUDA implementation gives the best result and remains almost constant. This is because this section of the code was completely parallelisable. We can also see that Open MPI performed well having also beaten the sequential time.

Varying Dimensions: Finally, for varying the number of dimensions (Fig.7), the CUDA implementation was the worst due to the data contention whilst Open MPI performed the best. This is a similar scenario to that of K-Means clustering as discussed above.

Effect of Processors It can be seen in Fig.8, that by increasing the number of processors we nearly half the computation time. This is highly desirable. This shows that the problem exhibits a high degree of parallelism and benefits greatly from parallel programming methods. Thus using Open MPI with a maximal number of cores for the data set should always be the goal.

Limitations & Difficulties Encountered

Floating Points: We encountered issues with errors compounding fairly quickly when there are a large number of iterations with the random values being in the range of 10^3-10^7. This makes sense given the extensive use of square roots and power functions in each algorithm. The errors are typically fairly small, often being quite closely related to the chosen maximum value allowed. Thus for very close approximations, our parallel methods work.

CUDA Memory: We made extensive use of global memory. This lead to a decrease in performance. Given that the calculations for our observations would have at minimum 2^{18} observations, this very quickly leads to bandwidth becoming an issue. This could be improved by storing components of each in constant memory such as; the observation's position & cluster's association. This would greatly reduce the bandwidth used. This could be further optimised to make use of shared memory, which would produce optimal results.

Processes Open MPI's performance is directly related to the number of processes that can be spawned. If you have a limited number of processors then Open MPI's results will not generalise. Thus there would be a hard cap on the performance gain that is possible.

Conclusion

Clearly, it can be seen that parallel methods can be used to dramatically decrease computation time in Fuzzy C-Means & K-Means clustering. Overall the best approach to use is Open MPI. This produced the best results. When there are large numbers of clusters CUDA is also a desirable method. Should the memory be changed for CUDA, this might lead to significantly better performance. Also, a hybrid implementation of the two methods might also generate the best results.

References

Dunn, J. (1973). A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters. *Journal of Cybernetics*, 3(3):32–57.

Macqueen, J. (1967). Some methods for classification and analysis of multivariate observations. *Berkeley Symposium on Mathematical Statistics and Probability*, 5(1):281–297.