Distributed systems final project - Distributed k means algorithm using Mpich2, Omp & Cuda api

Ruslan Abramov id 306847393

Implementation notes: - this file is the same as my readme file just easier to read

1. While studying how to implement k-means algorithm I encountered some theories on how improving the clusters convergence, most of these theories stated that choosing the first k clusters might affect greatly on the number of iterations it will take in order to get as close as possible to convergence – therefore I have decided to implement one of these theories.

The way that I set the first K clusters:

1. I choose a random point as the first cluster
2. Now I evaluate the center mass of the clusters
3. The next cluster will be the far most point from the center mass
4. Repeat

2. My k-means algorithm work in such way:

a. The mater sends the slaves all the clusters

b. Each process evaluates to which cluster each one of his points belongs

c. Each process creates a temporary cluster that contains it's accumulated X & Y axis coordinates and the number of points in it

d. Each process calculates how many of its points switched clusters

e. Each slave sends the master how many points switched -> the master calculates how many points switched cluster and sends this back to slaves

f. In case points switched clusters:

The slaves send their temporary cluster to the master

The master combines the temporary cluster data and calculates the new cluster

We move back to step a

g. in case points didn't switch clusters:

The slaves send their temporary cluster to the master

The slaves send their points to the master

The master combines the temporary cluster data and calculates the new cluster coordinates

The master calculates the diameter of each cluster

The master calculates the cluster quality if the quality is better than the one we need we finish

If the quality is not good enough each process moves his points in time and we go back to step a

3. My parallelism:

a. I have decided to split the points between the processes, I do it using mpi – making each process working with n/process,

b. I have decided to use cuda for 4 operations:

1. I calculate the distance of a point from the center of mass – each point is individual so this is a classic case for parallelism

2. I calculate to which cluster each point belongs – each point is individual so this is a classic case for parallelism

3. I add the accumulated X & Y axis coordinates of each cluster (this is done using atom functions because it is a critical area) – each point is individual so this is a classic case for parallelism but clusters are shared hence the use of atomic operations

4. I calculate each point's far most distance between another points in the same cluster – this found out to be not as quick as I expected on the contrary this was really slow – I have decided to use omp for that

5. I move points in time – each point is individual so this is a classic case for parallelism

c. I must admit that I didn't used omp much expect for b.4, due to the fact that most of for loops had critical areas.

4. Conclusions:

This project was more challenging than I expected, I must say that my results were ok, but the upsetting part is that it is not fast enough I suspect that my ankle point is the calculation of the distance between each point in the same cluster, I was sure that doing it with one process in cuda will provide much better results, but I was proven wrong, it is too late for me to modify this, but I have an idea , if each process will have all the points and each process will calculate the distance between N/process and all other points in the same cluster instead of one process calculating distance between N points and all other points in the same cluster the time complexity should be improved in a factor of 1/Number Of Process, which in our case is 1/3 –

I found a way to improve this using omp but it still doesn't make sense to me.

As for complexity – k means is np hard but I can say that each iteration is O(n^2) at most.

As for complexity – k means is np hard but I can say that each iteration is O(n^2) at most.

Some additional notes:

1. I found 1 bug that I still haven't had the time to fix - the bug occurs upon creating two different clusters at the same point, when we calculate the quality we divide by zero and by that the program starts to misbehave.
2. I found out that moving the points in time doesn't really effects the quality I have a theory for that, the files which I was using all have random set of points and by moving these points all in random speed will not make much of a change. I find sense in that, but maybe I am wrong.