Parallel implementation of K Means using MPI,OpenMP and CUDA

Introduction system

The system consists of number of fixed circles in area of width x height pixels.

Each circle described with canter coordinates and radius.

The circles are given in input file named circles\_x\_y.txt where x is the number of circles and y is the range of radius range for this set of circles.

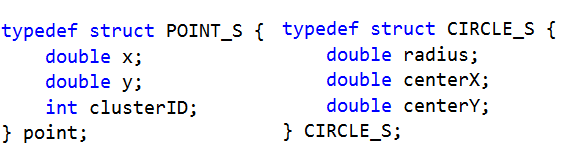
The system state is changed in time (T), where each different delta T time (t) there are moving points (p) around circles radius described before.

Each point moves around its center (cX, cY) as follows:

pX = cX + R \* cos(2\*Pi\*t/T)

pY = cY + R \* sin(2\*Pi\*t /T)

Points And Circles are saved in structures:



Introduction program

Due to system state changes there are different clusters of points created in the system,

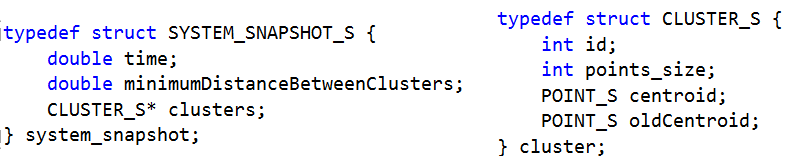
The goal of the program is to detect this clusters each system state change and finally

Find the minimum distance occurred between 2 clusters in all system state changes.

For doing so I using K Means algorithm which is designed for this type of work.

Each delta t I record system clusters and save snapshot of current state.

Clusters And Snapshots are saved in structures:



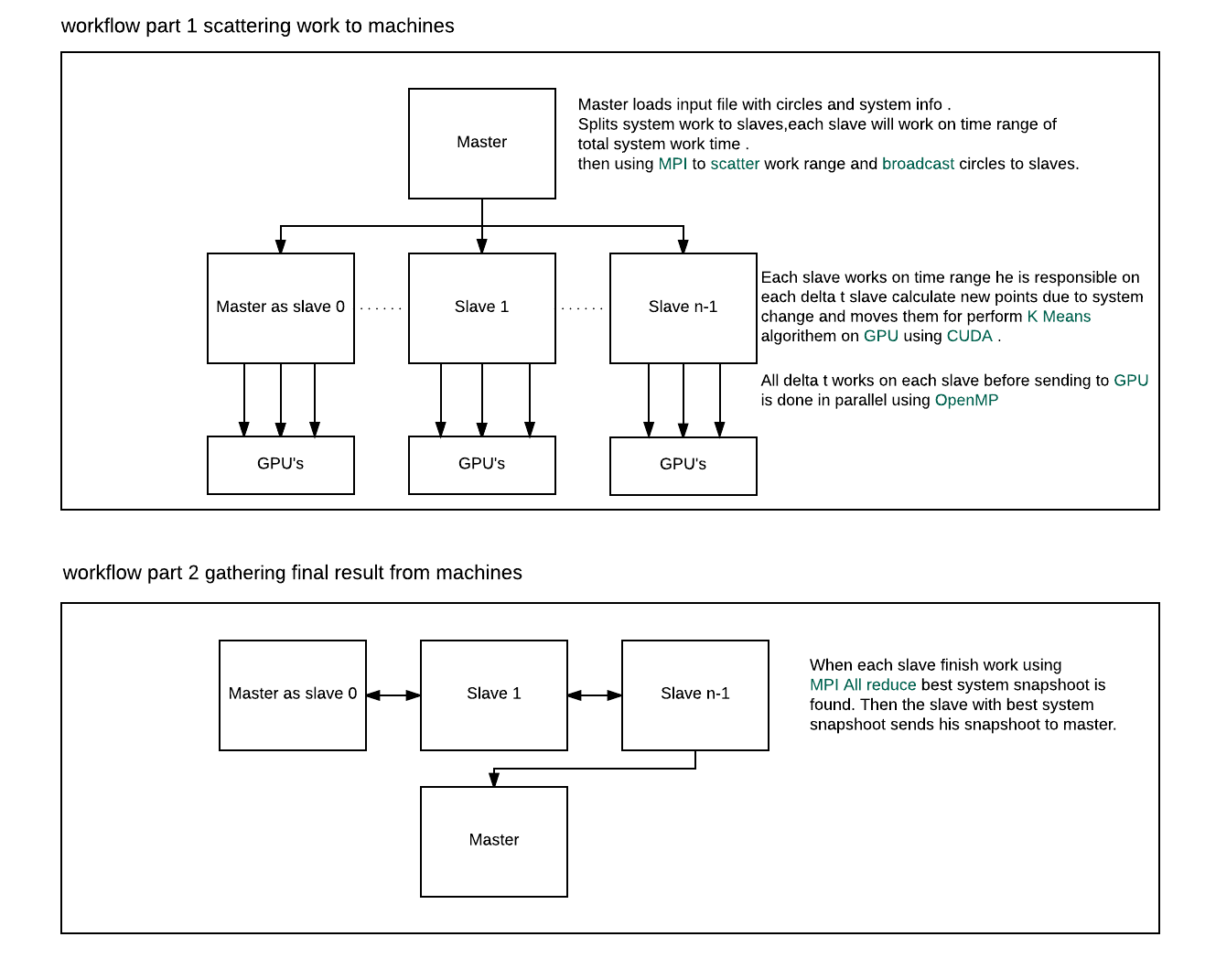
The goal is the find the best system snapshot between all snapshots which is the one with the minimum distance between clusters.

Combination of parallel technology’s:

The system consists of big number of circles which produce big number of points which involve big system changes, for getting results efficiently and in short time I have to use tools for shorten time calculations .

In this project I choose to use MPI technology for separating the work between different computers on network, OpenMP for running small tasks in parallel on CPU

And CUDA for running big tasks in parallel on graphics card GPU.

Program Work Flow Described by UML and Text:

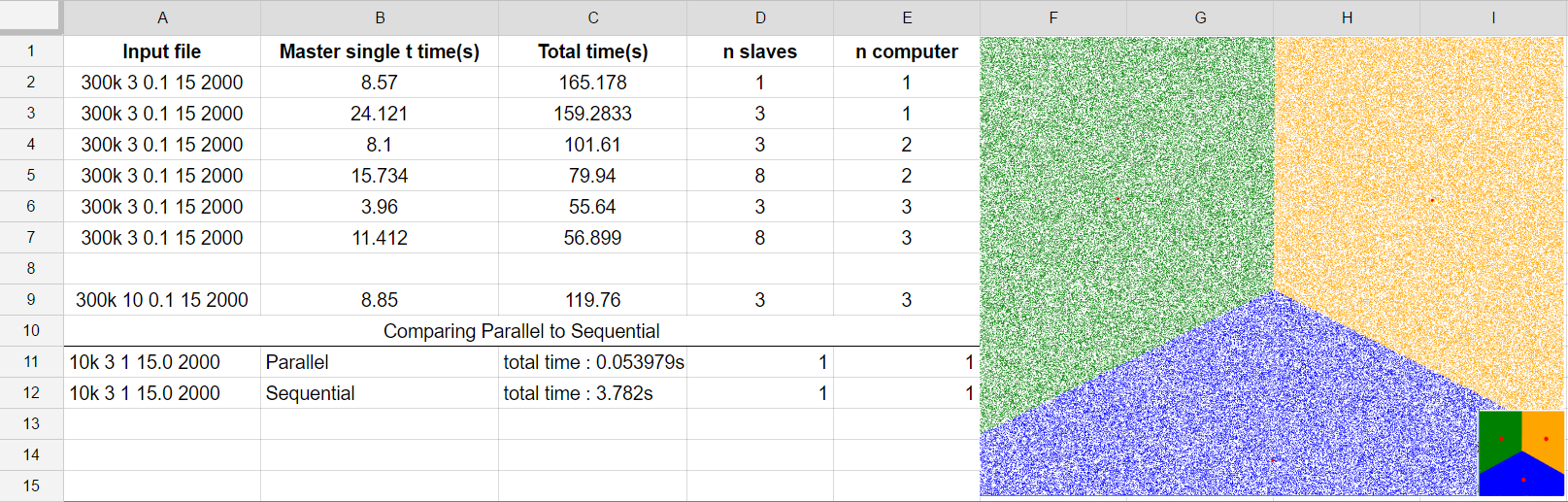
Depends on number

of cpu’s on slave machine

Results and Time testing example:

In this test I used input file with 300,000 circles and searched for best system snapshot between

150 system snapshots (total time system work 15s delta t 0.1s) the number of clusters to find is 3.



Test conclusions

As you can see the best result is found on row number 6 when I used 3 computers where each computer where running exactly 1 slave.

Iit turns out that when running more than one slave on same GPU using MPI it takes more time for program to finish due to network manage time and queue for accesses number of processes on same GPU.

Also as you can see in row 9 the number of clusters to find is increased to 10 from 3 which caused total time to increase from 56s to 119s means the number of clusters affects exponential time increase.

Complexity evaluation for single slave

nc = number of circles

np = number of points

nc=np=n

ncs= number of clusters

nt = number of system snapshots to calculate ( number of delta Ts)

nki = max number of k means iterations ( K Means limit )

nb = number of blocks on GPU

nt = number of threads per block on GPU

ntgpu= nb\*nt number of threads on GPU

ncpu = number of cpu on slave machine

d= number of mathematical operations

Work on CPU:

calculate points for current delta t using circles with OpenMP O(n\* 2d/ ncpu)

send work to GPU using OpenMP O(nt/ncpu)

perform K Means iteration O(nki)

Work on CUDA :

each thread calculate distance from point to cluster centroid using CUDA O(n\*6d/ntgpu)

each thread assign point to closest cluster O(n\*ncs\*6d/ntgpu)

Work on CPU:

calculate new cluster centroids :

-sum for each cluster all points x,y and number of points O(n\*3d)

-average new clusters centroids using OpenMP O(ncs/ncpu)

find minimum distance between clusters using OpenMP O(ncs^2 \* 6d/ ncpu)

As you can see the main load is in the summation for clusters points x, y variables in order to calculate new cluster centroids O (n).

This part is very hard to be fully parallelized because summation operation should sum many data to same memory address which can cause write conflicts.

One of the solutions I have read on academic article is to use sum reduction for making the summation operation, points are separated to blocks on GPU and each block calculate his part of sum, then on the CPU main summation is done.