Lab 4

pans dengbow

In this project, we implemented a parallelized version of the K-Means data cluster algorithm using OpenMPI for Java and a sequential version to compare with.

Parallelized K-Means Algorithm:

We a master and several slaves. They are processors.

First, randomly pick up centroids from generated dataset, then allocates each data to a processor by comparing the distance which is defined based on the data type with the centroid. After that, find the average data point as a new centroid, check for convergence based on the distance from the old centroid. If the distance is bigger than the threshold which is defined by user, do the above operation except for the initialization in a loop until no centroids move more than the threshold.

In the above process, the dataset is partitioned by the number of processors and each process have their own dataset. The master does the initialization and distributes dataset. The slaves do the allocation of data, then the master collects all data and does the checking and repeats. After all, the parallelization part is data allocation.

We also has a sequential version of K-Means algorithm as a baseline for comparison which simply runs on one processor.

Data Interface:

We provide a data interface which has a distance method and a average method. The distance method calculates the distance between different data points, and the average method return a data point in the central position.

DNA data:

DNA data has a string of DNA cases as a strand. The length of the strand is defined by the generator.The distance function for DNA was simply a count of the number of bases which were different. The averaging function for DNA was just the most common base in each position.

Point data:

Each point is within a range which is defined by the generator. It has an int value x and an int value y. The distance is mean squared distance, and the average data point is just the central point.

Data Generator:

For DNA data type, it generator strands of DNA data. the number and length are defined by users.

For Point data type, it generator a number of points within given range.

Compiling:

Navigate into the src, and run make to compile all code and make clean to delete all generated files.

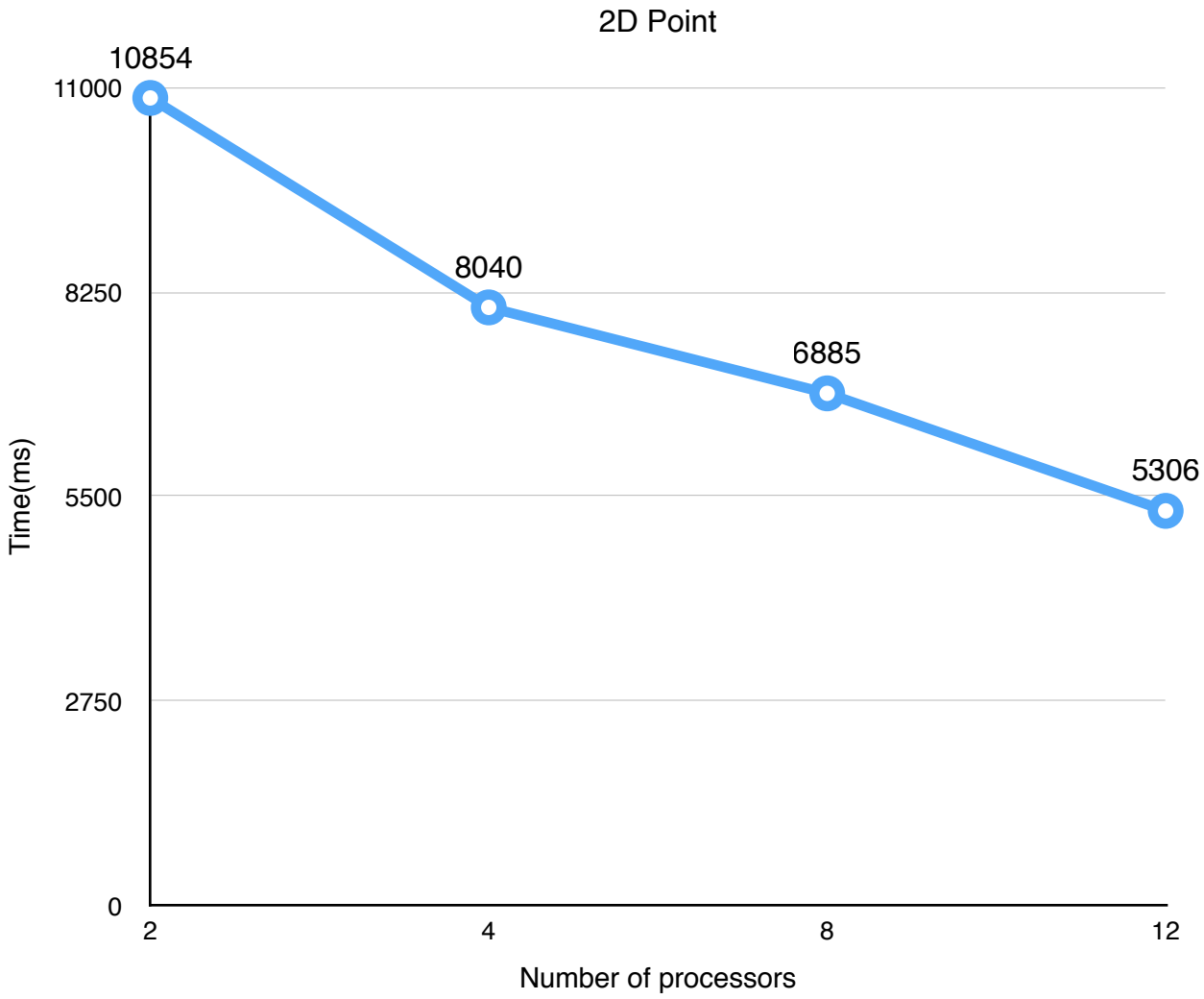
Running:

First, you need to run the ghc.sh to add ghc clusters to your host file. then just simply run the .sh file to run the code. You can adjust the data point numbers and length and range in these files, also the number of processors, cluster name, threshold. and number of centroids.

Analysis:

2D Point:

We randomly selected 1,000,000 2D points ranged from +1000 to -1000 as our dataset. The axis value is Int. We randomly choose 4 points as new centroids and set the distance threshold 1, then begin the iterations.
Sequential K-Means clustering used about 4000ms. The outcomes are shown in below figure.

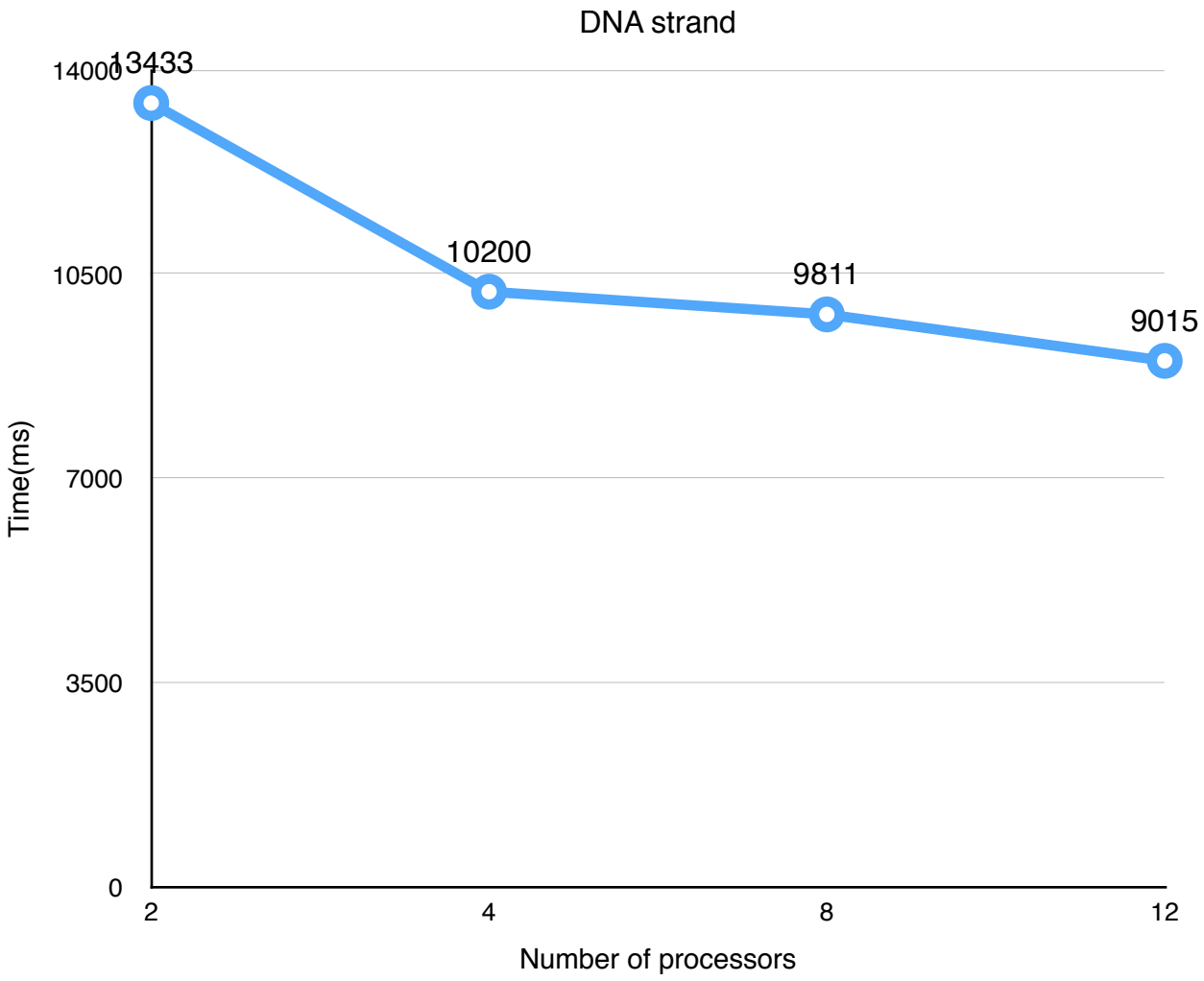


Compared to sequential version, parallel version takes more time even using 12 processors. This is due to communication overhead. Besides, the computation time decreases as the number of processors goes up. In this case, we can say that parallelizing the K-Means algorithm improves performance.
To calculate the running time, we run each test several times to calculate the average as our ultimate result. This is due to the fact that we select centroids randomly.

DNA:

We randomly generated 1,000,000 strands of DNA bases, each has 100 bases. We randomly choose 4 points as new centroids and set the distance threshold 1, then begin the iterations.

Sequential K-Means clustering used about 10000ms. The outcomes are shown in below figure.



From the above chart, we can see that DNA K-Means algorithm is similar to 2D point. However, the sequential version of 2D point perform much better probably because of the complexity of computation since DNA version takes string value not int value.
Running on 4 processors could be a sweep spot. Time decreases a lot here.