15–440/640: Distributed System
Lab 4 Parallel Kmeans

Name: Zeyuan Li, Guanyu Wang

1 Approaches to parallel the program

The parallel version of k-means using MPI is to initiate multiple processes based on a single piece of code. The pseudo code is like follows:

(1) Devide all data into equal pieces by rank=0 process and MPI_Send to all other processes

(2) MPI_Bcast initial cluster centroids and dimensions

(3) For each process, assign labels to each of the data points (find membership)

(4) MPI_Allreduce the sum of each element in the cluster centroid and cluster size and for all processes to get the total sum of each element in vector of cluster centroid and total cluster size

(5) For each process, calculate the new cluster centroid.

(6) Repeat (3)-(5) until converge

(7) MPI_Reduce the running time for k-means and for the subsystem as a whole to find the longest time as the system k-means and system total time respectively.

1.1 Parallelize main driver

The main driver program running on each process calls kmeans_read() to get the its subset of input data. Then $rank = 0$ process broadcast the initial cluster centroids to all other processes. This is followed by the main kmeans() on each processes. Finally, we do some clean up and reduce to get the longest time as the whole system running time.

1.2 Parallelize read input data

The kmeans_read() step is different on $rank = 0$ process and all other processes. The $rank = 0$ process read the whole input file and divide the input as near equal sizes. Then $rank = 0$ process uses MPI_Send to send each chunk of data to respective process. The all other processes just wait their and use MPI_Recv to get the input data for them to process.

1.3 Parallelize k-means

The parallel kmeans() function is as follows:

(1) Every processor use MPI_Allreduce to sum up local number of data to get the total number of input data points.

(2) Computing and recording the cluster membership for local data points.

  (a) For 2D data point, choose the nearest cluster center (Euclidean distance).

  (b) For DNA strand data, choose the cluster center which has the largest similarity (as described in the write-up).

(3) Changing the temporary clusters locally and also record the changes of membership locally.

  (a) For 2D data point, add the data points' cooridinate values to a temporary record, and also update the number of data points in every cluster.

  (b) For DNA strand data, count the number of appearances of every DNA symbol in every position.

(4) Every Processor use MPI_Allreduce to sum up the temporary record and cluster sizes (for 2D data points) or the counters (for DNA data).

(5) Updating the new cluster centroid.

  (a) For 2D data point, for every cluster, the new centroid is the average value of every coordinates (divide the sum by its size).

  (b) For DNA strand data, the centroid DNA strand is constructed by the most frequent symbols appeared at every position in the same cluster.

(6) Every Processor use MPI_Allreduce to sum up the total changes in this iteration. If the changes is smaller than the predefined threshold (or the iterations is larger than the predefined maximum iteration number), then stop, or repeat from (2).

1.4 Parallelize write output data

The kmeans_write() step is different on $rank = 0$ processor from all other processes. The $rank = 0$ processor receive the memberships of all data from other processors using MPI and output them together into the output file.

2 Experimentation and analysis

Please refer to Fig. 1 and Fig. 2.

Note here the all processes time is the sum of all proc time, it is easy to see that the average time for evey processor decreases when the number of processors increases.

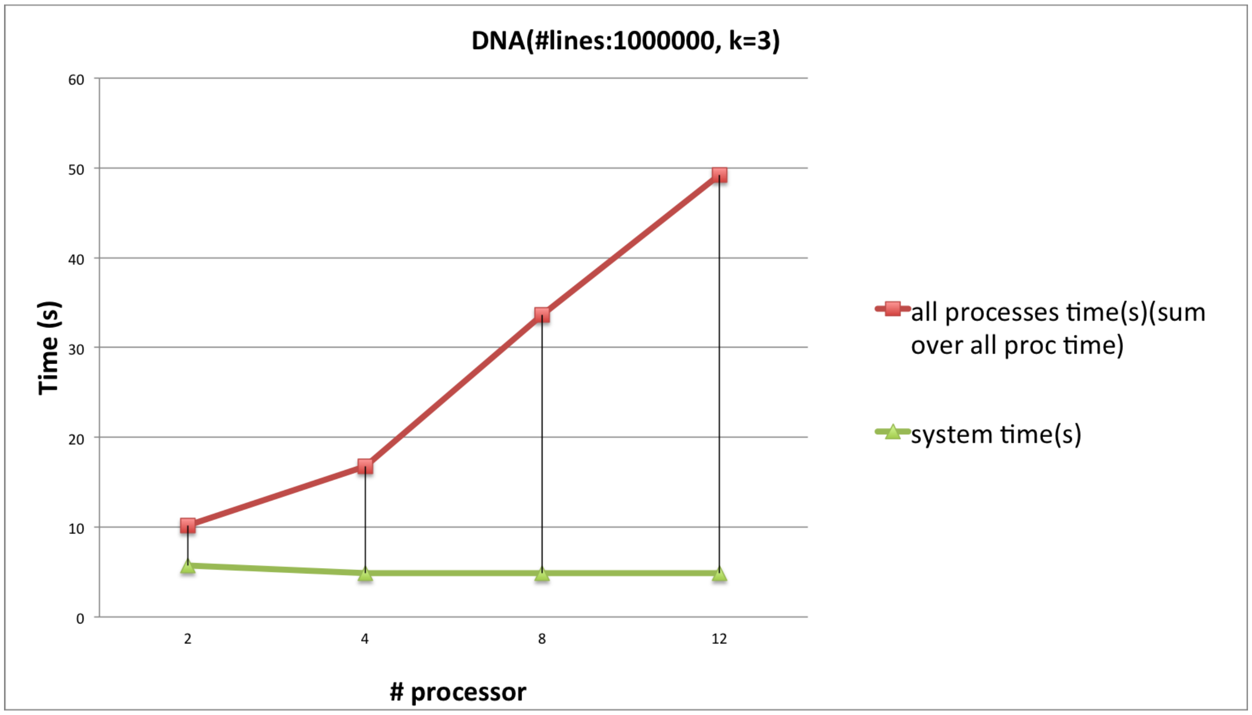

Figure 1: Plot for the time with DNA data

According to our experiment, we can find that the parallel k-means methods need less time when the number of processors increases (i.e. the degree of parallism increases). And the system time shows that for the whole system, the time for the task actually does not decrease a lot even the number of processors increases (6x more processors when there are 12 processors). This also can be explained a little by the following fomular (but it is far from

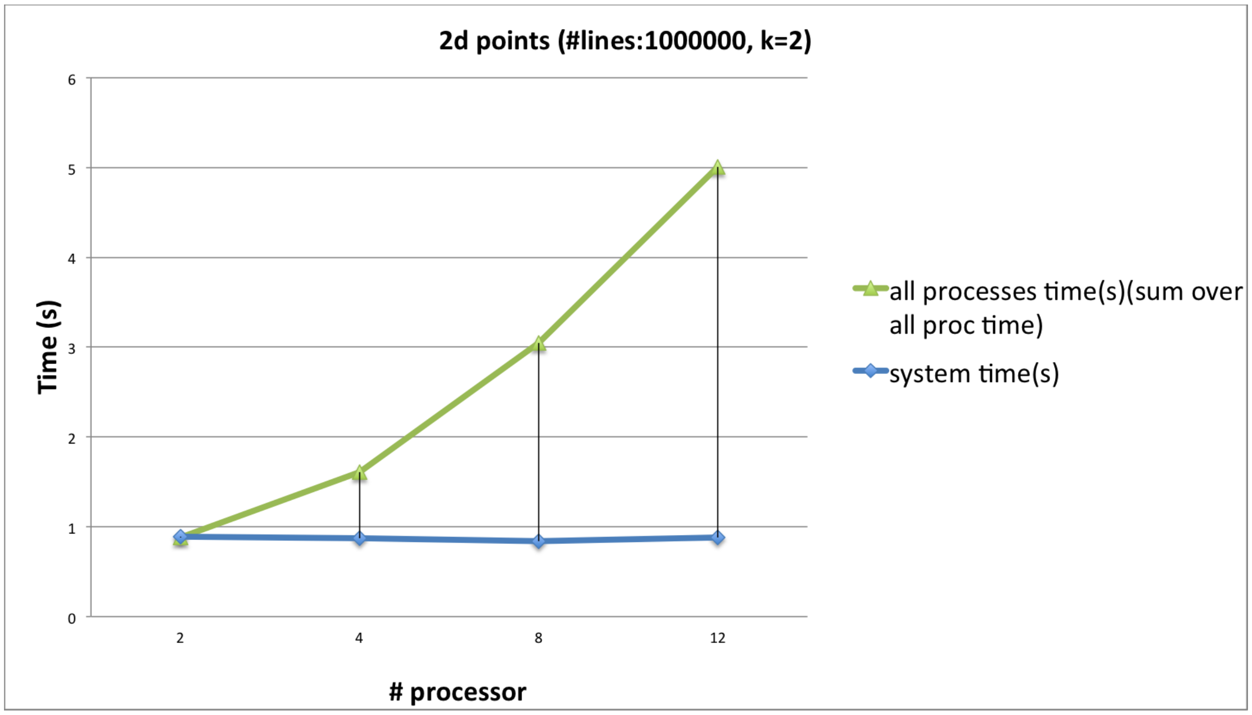

Figure 2: Plot for the time with 2D data

the ideal case described by this formula):

$$speedup = \frac{1}{s + \frac{1-s}{p}} \tag{1}$$

3 Usage of our code

(tested on GHC machines)

(1) Using ./run.sh or ./runDNA.sh to generate 2D data or DNA data (the parameters can be adjusted in the shell script, please refer to the explaination in the scripts).

(2) Using "make" to make all source codes.

(3) For 2D data demo, using "make ghc" run the k-means on 2D input file "data/input/cluster.csv" . For DNA data demo, using "make ghcdna" to run the k-means on DNA input file "clusterDNA.csv".