Parallel Final

Jennifer Kulich

December 7 2020

1 Introduction

This program will use the k-nn algorithm to find either the classification of a data set based on a given query using MPI. In this program, the user will input the data set and query, and the program will find the classification. It should be noted that I ran into some issues, and the program doesn't necessarily work the way it should. While programming, I got to the point where I couldn't declare the variables I needed to finish, and I think that was because of the number of 2D arrays I was creating. After doing some research, I really should have just created a 1D array and dealt with the weird math that came along with indexing it. I ran out of the time to do the conversion, but so my program *partially* works. By partially, I mean that if you only set k to 1, the program will run correctly. If you put in more than 1, the program will still run, but it may not print out the right classification. If I had the time to do the conversion, I would have found how many types of classifications there were, created an array to hold the 'count' of the classifications as I found the minimums. Then, I would have printed out the one that has the most. I do have code for this commented out.

1.1 How to Run

I have provided a makefile to help run the program. You can do 'make clean;make all". Then you can run: mpiexec ./knn k data set query set . As explained above, a k of 1 is really the only thing that works properly, but you can run it with a k greater than 1.
Please make sure to only have a query with the same amount of data points as the data set (minus the classification)! For example, the iris data set as 4 data points and a classification. This means that there should only be 4 data points in the query.
Another side note: every once in a while, there seems to be an issue cleaning, making, and attempting to run the file. This happens maybe every 30 or so runs, but it's usually not any major issue. If you have this happen, just run the program again, and everything should be fine.

1.2 Expected Format of Data

The data should be in the format of number,number,number.... with a newline to break off the data. The data may be an int, or double. The classification should be converted to an int or double. The program will not work any other way. I have provided the iris data set.

1.3 Expected Format of Query

The data should be in the format of number, number, number... The query should only be one line long. More than one query is not accepted. I have provided a query where the query is the first point in the data set. It should return a classification of 1.

1.4 Outline of the program

The program utilizes MPI, so it will include the mpi.h library at the very top. All other libraries included are the typical standard libraries, and the math library.
The program first starts with reading k and the two files provided in the command line. It will then open the data set file, find out how many lines and data per line are in the file, allocate a 2D array and read in the contents of the file, adding an extra column so the distances can be calculated. From there, they query file will be opened, a 1D array is allocated, and the file is read in. From there, the number of data lines, amount of data per line, and the data sent and query are sent to their respective tasks (it should be split evenly among each task).
In each task, the information will be received, 2D or 1D arrays will be made and filled. From there, I would have liked to have created a 2D array to hold just the distances and classifications, but that wasn't possible as described above, so the code has been commented out. The distances were found using the distance formula and placed in the last column of the 2D array. From there, I would have created an array to hold the count of the classifications that could be used once the data was sorted and the k smallest distances were found, but that too was commented out. From there, I find the smallest distance, record the classification and do an MPI_Reduce with a minimum reduction and send that number to task 0. If I would have been able to do things the way I wanted to, I would have I would have found the minimum classification amongst all tasks and then recorded that classification instead of sending it to task 0. Only once I had done this k times would I have found the mode and sent it to task 0. Then in task 0, I printed out the classification.
Right now, the program is broken so if you put in k = 1, you will get the correct result. If you put in k=2, you will still the same answer as k = 1. This is because of how things were structured. If I had done it the way I wanted to, I would have had a sorted list, and all I would to have done was index k elements into the sorted list to get the smallest number. Again, this is what I would have

liked to have done but couldn't.

1.5 Data Analysis

Well, based off of what I have, I couldn't do much data analysis. Even if I could have, I wasn't able to because I wasn't able to declare the variables I needed.

1.6 Fosters Design Methodology

1.6.1 Partitioning

The partitioning would be calculating the distance from the query, sorting the data based on the query, and finding the classification based on the input k (finding the mode).
These are probably the most basic tasks of the program which is the goal of partitioning. It avoids redundant computation, and the tasks are relatively the same size (sorting the data may be a little bit different.

1.6.2 Communication

The communication needs to be between the finding the distance and the sorting since the sorting needs the distance of each data point. There also needs to be communication between the sorted list and the data points to be able to find the classification. There also needs to be communication between the sorted list and finding the classification- we needs to know the smallest distances to be able to get the first k classifications with the smallest distance from the query. Each task does perform about the same amount of communication and can be run concurrently.

1.6.3 Agglomeration

Agglomeration is done to reduce the communication overhead in a given process. The finding of the distance and sorting can be grouped together since they depend on each other. This reduces the amount of communication, avoids replication of data, and the computation and communication are about the same of every task.
Now that's not to say that it cannot be done any differently. The finding of the distance could be done on multiple tasks before sending all of the information back to task 0 where the sorting is all done in a parallel way (potentially OpenMP), and the classification is done using the first k elements in the sorted data set. I was not able to implement this, so I cannot say if it would be faster or not, but I have a feeling that it is not faster.

1.6.4 Mapping

The data should be evenly divided onto different tasks. I don't think it could be done much differently, and it would work very well considering the data set will most likely be large and there should be some form of balancing between tasks.