**15-440 Distributed Systems**

**Project 3**

**Hira Yasin Dhamyal**

**Experimentation and Analysis**

*Comparison between 2 different K-Means implementations in terms of :*

*Development Effort:*

The algorithm for K-Means for 2D points was easy to understand and coding did not take much time, except for a little problems here and there. For 2D points, deciding what the new centroids will be was easy and did not take much time. Also it was easy to understand what work will the slaves do and what will they send back to the master.

On the other hand, the algorithm for DNA clustering was a bit tricky. Where deciding how to calculate the new centroid was tricky.

For both of them, the initial centroids were chosen depending on the number of clusters given and then for each cluster taking the middle element. For example, there are 400 points in the data set and 4 clusters. Then each centroid will be the 50th, 150th, 250th, and 350th element.

*Performance:*

The MPI Implementation of 2D points is very fast in contrast to the DNA Strands. I think that is because in DNA Strands, I am using MPI\_Send for every character, which makes it pretty slow.

*Experience in applying MPI to the K-means clustering algorithm:*

The experience was tough. It took me a lot of time to ensure the Sends and Receives were coordinated together, meaning that there were equal number of Sends and Receives and also every Correct Send was related with the correct Receive, by using the same tags. In MPI implementation for 2D points, the sends were only for the integers or integer arrays, which was a lot easier to handle than the send and receive of the char arrays. In MPI implementation of DNA, I tried a lot to send the whole string arrays, but it did not work, so I had to resort to the idea of sending every character one by one, which is why my MPI\_DNA is taking more time than even the sequential, which is sad I think.

*Insights concerning the performance trade-offs of MPI and sequential programming with K-Means:*

Personally, programming the sequential was very easy than the MPI version. But I guess if I was used to coding in MPI, it would have been easier. You have to spend some time deciding what work will the slaves do and what will the master do. The time that the MPI version takes is less than the sequential version. But the thought process is more for MPI.

*Thoughts on the applicability of K-Means to MPI:*

The applications of K-Means algorithm by itself are a lot and we just did 2 of them. A little search shows that it is very important for Search engines and in wireless sensor networks. So it seems very important in practical applications. Using MPI makes the K-Means algorithm fast on large data sets which means that in practical applications, the algorithm can provide the results in a short time.

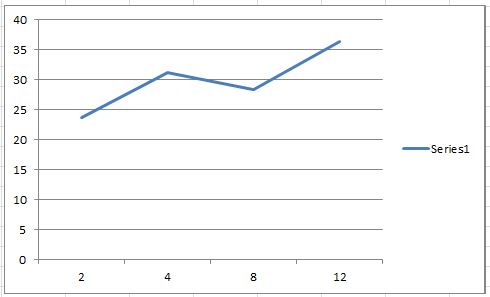
*Recommendations regarding the usage of MPI for algorithms similar to K-means:*

MPI could be used for any algorithm which involves large datasets and where work can be divided to other processors. Like the frequency of words in a file.

*Scalability Studies*

1. *Number of Data Sets: 1000000*

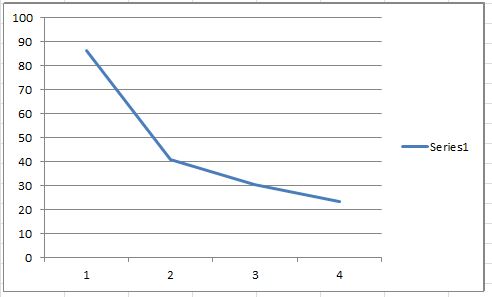
|  |  |
| --- | --- |
| No of Virtual Machines = 4 |  |
| No. of processors | Time (sec) |
| 2 | 23.708212 |
| 4 | 31.248183 |
| 8 | 28.446121 |
| 12 | 36.380325 |

****

For the number (1) experiment, as the number of processors increased, the time overall increased. I think as the number of processors keep on increasing, the time will increase as well. But when there were 8 processors, the time decreased for when there were 4 processors. But the time was not lower than what it was for with 2 processors. So the lower the number of processors, the less the time.

*(2) Number of Data Sets: 1000000*

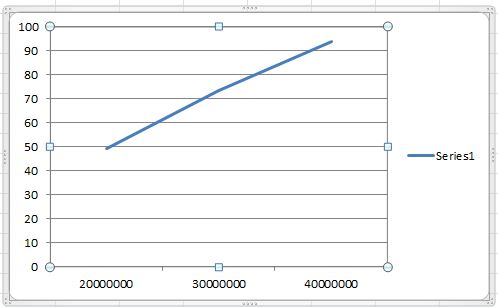
|  |  |
| --- | --- |
| number of processors = 4 |  |
|  |  |
| No of virtual machines: | Time (sec) |
| 1 | 86.354169 |
| 2 | 40.977728 |
| 3 | 30.487959 |
| 4 | 23.265849 |

****

With the number of processors fixed at 4, the increased number of virtual machines decreased the time of the K-Means by a lot. But it seems like as the number of virtual machines increased, the speedup we can get is decreasing. Which is exactly what we studied in class; that there is a limit to which the increased number of machines can decrease the time of the program (Amdahl’s Law).

*(3)*

|  |  |
| --- | --- |
| number of processors = 2 |  |
| number of Virtual Machines = 4 |  |
|  |  |
| Data Set | Time |
| 20000000 | 49.137027 |
| 30000000 | 73.568428 |
| 40000000 | 93.700049 |

****

With the data set increasing, the time for each increased linearly. Which means that the time for K-Means is linear with respect to the input size ; here the input size is the number of 2D points. I believe as we keep on increasing the size, the time will always be linearly increasing. (With the number of processors and virtual machines fixed).

It would be interesting to find the best number of processors, the number of virtual machines, and the size of the data set which make time the lowest.