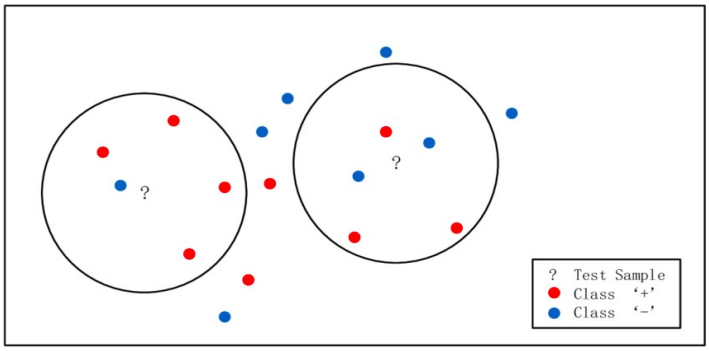
Parallel and Distributed Systems
Assignment 2 – KNN Ring (open MPI)

- **Authors:** *Portokalidis Stavros AEM:9334 mail:stavport@auth.gr*
  Papadakis Charalampos AEM:9128 mail:papadakic@auth.gr



This is a report for importan obeservations and execution time results of the KNN ring problem using sequential and parallel programming (open MPI method) .
Source code can be found in the links below:

- https://www.dropbox.com/s/v8cqscgglnisxyo/code.tar.gz?dl=0
- https://github.com/sportokalidis/knn-ring-mpi

V0. Sequential

In the first part we implement the sequential version of the problem. The goal is to find the k nearest neighbors of each point in a given query set (Y) comparing with each point in a corpus set(X). For the implementation we use BLAS methods for quicker calculations of Euclidean distance between two set of points

V1. Synchronous

The second part is including parallel implementation of the problem using MPI (Message Passing Interface) ,to run as p MPI processes and find the all-KNN of a given set of points X .The Synchronous part defines that in this particular implementation there are used Blocking functions of the MPI environment (MPI_Send , MPI_Recv). In this version we are using the knn function that was created in V0. We move the data along a ring , so every process receives from the previous and sends to the next one. Every time the k nearest neighbors are updated . The goal is to keep in the final matrix the k nearest neighbors and the distances of each point comparing to the others.

- **To have all the points checked against each other , we need to iterate (p-1) times, where p is the processes we activate while calling the program.**

- **As you can see in the lasta page's table, the time spent in MPI communications is really short compared to the one we spend doing the right calculations for finding the right KNN neighbors.**

V2. Asynchronous

The third part of the assignment is a different implementation using non-blocking functions of the MPI environment (MPI_Isend , MPI_Irecv) . The goal of the asynchronous version is to hide the time that is spent between communications . The truth is we did not see much faster execution as you can see yourself in the last page's table. The knn ring problem is the same and the V0 knn function is also used in this part.

Global Reductions and all to all

For this part after computing the maximum and minimum distance that occurred after every process ran , we found the total/global maximum and minimum using reduction MPI commands (ex. MPI_Reduce).

Problem

The e-learning tester shows some error with undefined references according to the open MPI environment. However, when we run the program locally and even in the HPC the validation of the tester gives us back "CORRECT NEIGHBORS" every time. Not knowing that there will be extra time for the assignment, we submitted in the e-learning correctness check, a version that showed right only the sequential part and we could not check it online afterwards ,anymore.

Results

Below are results and graphs that occurred for the V1 and V2 parts. For running the program we used the HPC of Aristotle University of Thessaloniki (hpc.it.auth.gr) ,which provided the opportunity working with many nodes and processes.

Nodes & Processes per Node	Synchronous Execute Time(seconds)	Asynchronous Execute Time(seconds)	Synchronous Calculations Duration (seconds)	Synchronous Communications Duration(seconds)
2 Nodes / 1 ppn	10.665000	10.620000	10.475000	0.190000
2 Nodes / 2 ppn	5.817500	6.870000	5.612500	0.205000
2 Nodes / 4 ppn	3.280000	3.518750	3.175000	0.105000
2 Nodes / 6 ppn	2.420833	2.535000	2.315833	0.105000
2 Nodes / 8 ppn	1.941875	1.951250	1.874375	0.067500
4 Nodes / 1 ppn	5.670000	6.857500	5.460000	0.210000
4 Nodes / 2 ppn	3.117500	3.370000	2.992500	0.125000
4 Nodes / 4 ppn	1.885000	1.945625	1.818750	0.066250
4 Nodes / 6 ppn	1.407083	1.409583	1.342083	0.065000
4 Nodes / 8 ppn	1.095938	1.076563	1.040625	0.055313

N=12k , D=30 , K=150

Nodes & Processes per Node	Synchronous Execute Time(seconds)	Asynchronous Execute Time(seconds)	Synchronous Calculations Duration (seconds)	Synchronous Communications Duration(seconds)
2 Nodes / 1 ppn	55.005000	57.283000	55.005000	1.375000
2 Nodes / 2 ppn	30.205000	29.14000	29.157500	1.047500
2 Nodes / 4 ppn	16.601250	17.720000	16.005000	0.596250
2 Nodes / 6 ppn	11.672500	12.024167	11.080833	0.591667
2 Nodes / 8 ppn	9.017500	8.970625	8.650000	0.367500
4 Nodes / 1 ppn	28.900000	27.320000	27.820000	1.080000
4 Nodes / 2 ppn	15.655000	16.541250	14.988750	0.666250
4 Nodes / 4 ppn	8.708750	8.819375	8.819375	0.342500
4 Nodes / 6 ppn	6.372917	6.285000	6.285000	0.352083
4 Nodes / 8 ppn	5.119063	5.013437	5.013437	0.192812

N=28k , D=30 , K=150

Nodes & Processes per Node	Synchronous Execute Time(seconds)	Asynchronous Execute Time(seconds)	Synchronous Calculations Duration (seconds)	Synchronous Communications Duration(seconds)
2 Nodes / 1 ppn	260.810000	280.245000	260.810000	1.485000
2 Nodes / 2 ppn	96.347500	110.385000	92.355000	3.992500
2 Nodes / 4 ppn	48.428750	51.166250	46.106250	2.322500
2 Nodes / 6 ppn	32.954167	34.260833	31.350000	1.604167
2 Nodes / 8 ppn	26.267500	26.273125	25.219375	1.048125
4 Nodes / 1 ppn	82.417500	97.462500	78.775000	3.642500
4 Nodes / 2 ppn	45.521250	48.240000	43.083750	2.437500
4 Nodes / 4 ppn	25.288125	25.379375	24.120625	1.167500
4 Nodes / 6 ppn	17.048333	16.745833	16.112500	0.935833
4 Nodes / 8 ppn	13.659062	13.110625	12.809687	0.849375

Illustration 1: N=40k , D=30 , K=150