Neural Networks

Erika Koro

November 12, 2022

Objective

The objective of this assignment is to calculate the K-Nearest neighbours for all points of a dataset. Specifically, in the first serial source code is constructed the vantage point tree which sequentially is used in the KNN algortihm. In the KNNSearch the construction of the VP Tree per point is used in order to sort the array with the points, according to their median distance from the pivot and chooses the first-k elements as neighbours. In order to parallelize this algorithm, MPI specification for C is used to optimize performance and give a distributed solution. The code for this assignment can be found in here.

Experiments

The experiments were executed in an AMD Ryzen 7 5800H, 8 cores/ 16 threads, 3200.000 MHz. and the results are based on the number of points and dimension of the dataset and the number of searching neighbours is a power-of-two value with k = $2^{[1:8]}$.

Observations

First of all, by keeping stable the dimension and the number of processes (optimal=16) and increasing the number of points it causes 5 times slower KNN algortihm than MPI on average. Secondly, keeping the number of points, the dimension and the processes stable and increasing the number of k-nearest neighbours keeps MPI's execution time constant.Meanwhile, increasing the number of k increases KNN sequential's execution time. Also, increasing the number of dimension by a factor of two, caused 2-times slower performance in MPI. At last, as it is expected by raising the number of processes from 2 to 16 is observed the optimized performance, a number that can be predicted by the number of available cores.

Plots

Problems encountered

Initially, the most important issue I faced was debbuging the MPI code amd eliminating the memory leaks in order to give a bigger dataset but MPI's memory leaks filled my memory. Another problem I encountered, was trying to parallelize the sequential construction of VP tree by using the OpenCilk library but the execution time was equal to the sequential's. I also tried to give it a threshold size which optimized the parallel code for number of points / 4 and then the sequential code was executed. The reason why OpenCilk did not achieve better performance than the serial code was due to the workload per thread although the CILK WORKERS were set at the number of cores.