Parallel and Distributed Computer Systems

Second Assignment

Konstantinos Chatziantoniou 8941

–

Aristotle University of Thessaloniki

24/12/2018

Intro

In this assignement, we have to implement a distributed algorithm for creating a Vantage Point Tree for a dataset and then use the VPtree to do all-kNN search of the VPTree points, using MPI. MPI(Message Passing Interfeace) provides a high-level network abstraction,that is easy to understand, so that we can focus to the algorithm itself. MPI programs with multiple processes compared to serial programs, can be both faster, because of the parallelization, and slower because of the cost of the communications,that have a significant effect in speed.

Code

Code is available at github.

Vantage Point Tree Construction

Algorithm Explanation

At first, the processes will work together, in teams, to determine the nodes of the VPtree, up to the point each process has to work on a different node. Then, the processes will create the VP subtree in serial.
At the last level of the parallel construction, the number of nodes will equal to the number of processes. We know that the nodes of a binary tree, at a given level,assuming root node is at level 0, are $N_{nodes} = 2^{currentLevel}$ and in our case $N_{nodes@lastLevel} = 2^{lastLevel} = N_{procs}$. Therefore $lasLevel = log_2(N_{procs}) = totalIterations + 1$. The iterations stop after each node is created by a team of 2 processes and the 'parallel' iterations are $log_2(N_{procs}) - 1$.
To split the processes into teams, we will use the `MPI_Comm_Split()` function, dividing the global `processId` with the required number of processes in each team every iteration.

Steps at every iteration:

1. The master of the team(`group_rank` == 0) broadcasts the VantagePoint(VP) to the slaves.
2. Every process calculates the distance between the VP and their set of points.
3. Every process executes their part(MasterPart or SlavePart) and the master broadcasts the median to the slaves.(Master ans Slave part were modified to work with different communicator, `group_comm` instead of `MPI_COMM_WORLD`).
4. Every process partitions their set, so that the points at the left part of the array are nearer than median and at the right are farther.
5. Then every slave send to their master the `countEq`, `countMin`, `countMax` and the master orders the slaves to swap values equal to median for balancing.

6. The master determines how many points should everyone send and receive and messages to coordinate the transactions.
7. Slaves waiting for the master's message and proceed with the transaction as soon as the message arrives, until the messages tell them to stop.
8. At this point, the first half of the team has the nearest points, and the second half the farthest. Now, they merge the points received with their points, deleting the part that they don't need. Then they recalculate their `partLength` and reduce it (`MPI_Allreduce`) to find the set `size` of the team.

After the iteration ends, the global master gathers the part of the tree that each process knows and merge it to create the binary tree. For this tree(`final_mpi_tree`) the mathematical relation of the root and its children is

- Accessing the left child $LowChild_{index} = 2 \cdot Root_{index} + 1$
- Accessing the right child $HighChild_{index} = 2 \cdot Root_{index} + 2$
- Accessing the root from child $Root_{index} = (Child_{index} - (2 - Child_{index} mod 2))/2$

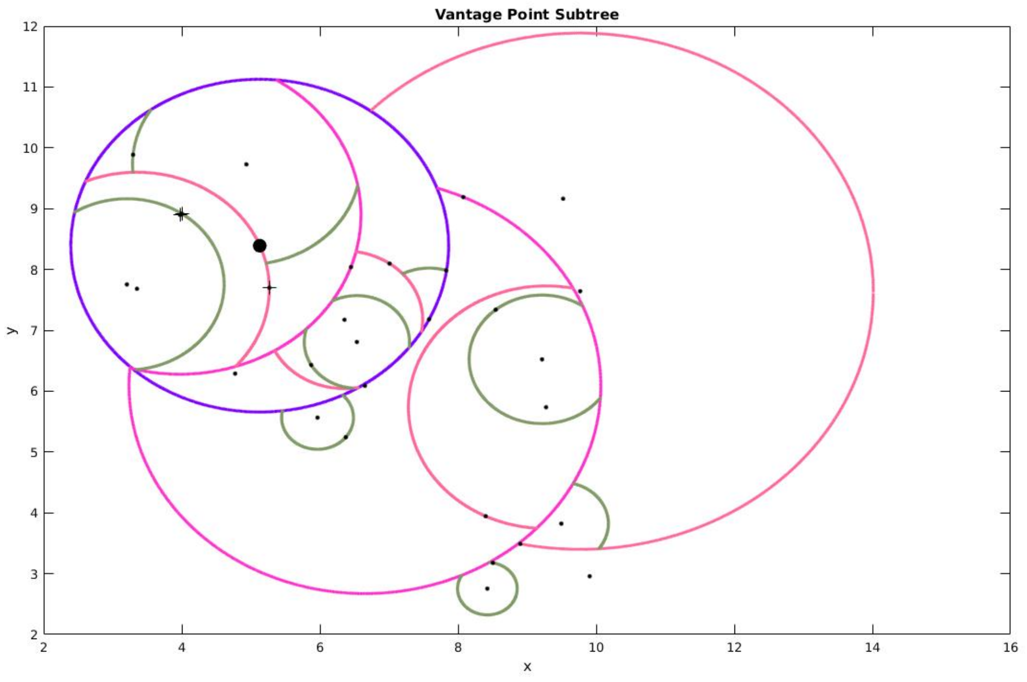
Then each process works in serial to construct their Vantage Point subtree. By recercively, choosing the first point as VP, point the array to the next point and partition it, we can create a binary tree in an '1D array'. For this tree(saved in `pointArr`) the mathematical relation of the root and its children is

- Accessing the left child $LowChild_{index} = Root_{index} + 1$
- Accessing the right child $HighChild_{index} = Root_{index} + 2^{maxLevel - currentLevel}$

Verification

After each iteration, we check that the points each process has(after the transactions and the merge) are lower/higher than the median (first half/second half of team), thus verifying the `final_mpi_tree`.

The correctness of the Vantage Point subtree is verified visually with a matlab script:



All k-NN

Local k-NN

First, each process finds the k-NN for their set locally. With the previous matlab script, we can verify the knn, not only visually but by finding the k-NN linearly too.

Global k-NN

After this, the processes have to search the `final_mpi_tree` and check if it's possible that another process has points that are nearer than the local kNN. The process enter a loop, until no one has to send points to others. The steps in the loop are:

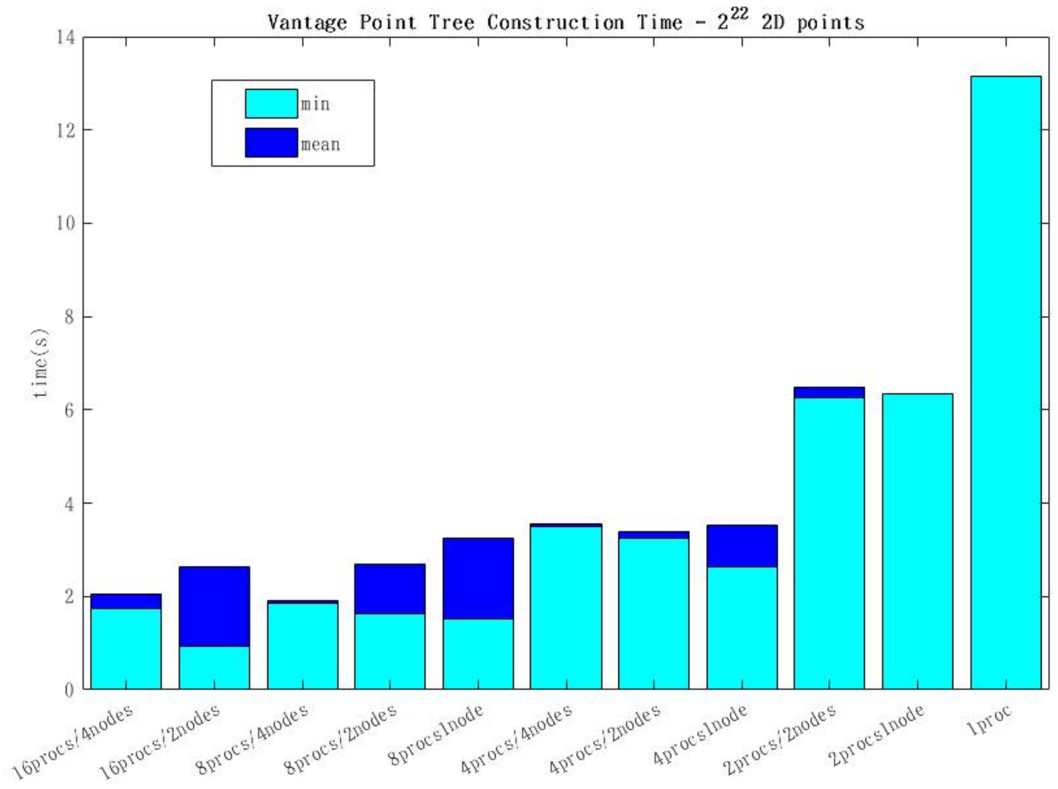
1. Check the `final_mpi_tree` to determine which process to ask for their kNN or if our kNN are ok.
2. If every processes' k-NN are ok, terminate the loop.
3. Count how many points to send to each process.S
4. Every process sends the count of points to send and then the points, while every other process waits to receive,starting from 0 process ID.
5. Every process finds their kNN for the points received.
6. Then the points are sent back to the origin and are checked with the previous neighbours, swapped if necessary.

Verification

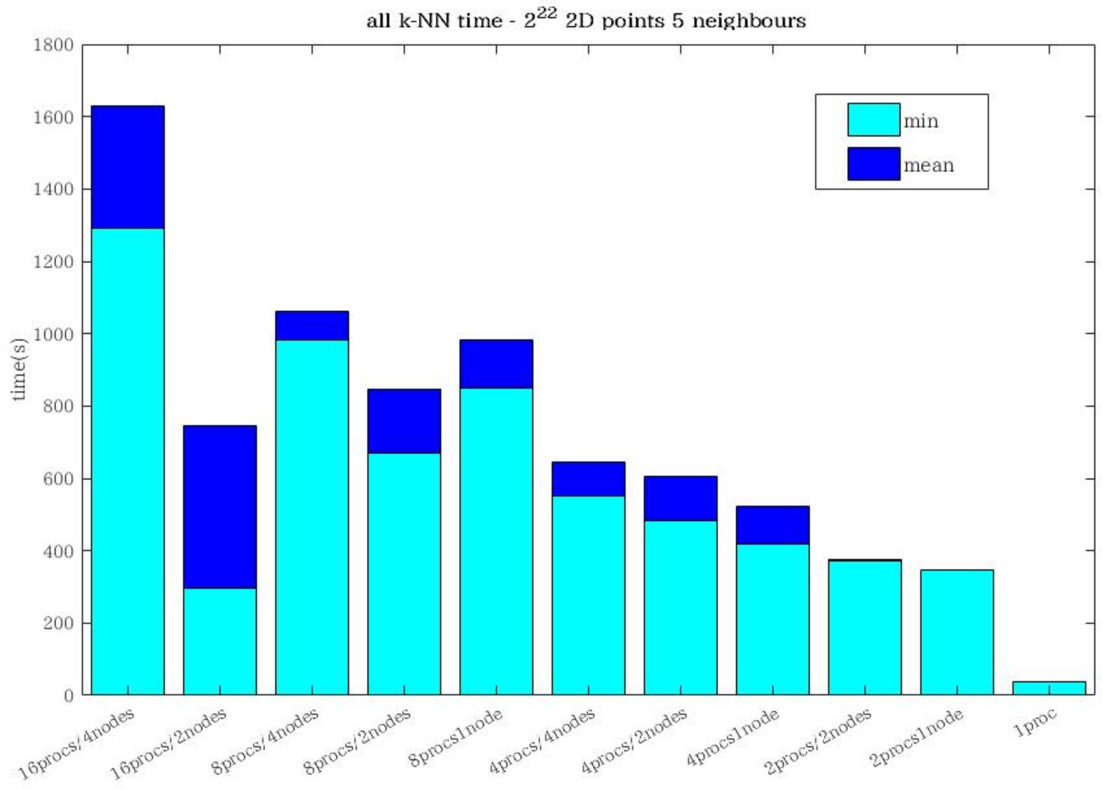
The results were 'verified' with the previous matlab script and plotting the points of the other processes.

Results

The following graphs show the time it took the programm to construct the VP tree and find all kNN for 2^{22} 2D points, with different amount of processes and nodes. The program was ran at hellasgrid infastructure.



As we can see, using more mpi processes reduces the time of the VPT construction. From 1 to 2 to 4 processes, time required is halved each step. From 4 to 8 procs, there is still a noticable reduction in time, but from 8 to 16 time remains almost the same. This kind of result was expected. Since we do not use any internal multithreading mechanism (openmpi,pthread…), 1 process uses 1 core for the whole tree. Increasing the processes, after the first $n_{procs} - 1$ tree-nodes, the tree construction is split between n_{procs} different cores or nodes and done in parallel. But, using more processes introduces more communications, that are 'expensive'. Thus, there is no remarkable difference in speed, between the 16 and 8 processes;the time we earn using more nodes/cores for tree construction is counterbalanced by the cost of communications. Also, there is no distinction between using different number of nodes for the same numberr of processes. Communications are only required for the first $n_{procs} - 1$ tree-nodes.



From this graph, we get the ‘opposite’ results. Using more processes for all k-NN is slower. Although the difference in time is surprisingly big at first glance,it can be explained. The more the processes, the more the subtrees that may include the neighbours of a query, leading to more communications. If we look back at the all k-NN search steps, we can see 2 major communications:

1. Sending queries
2. Sending k neighbours back for each query

There is no guarantee that one process will not send all of their points as queries, or that it will not need to ask every other process to search their subtree. Although,for more than one process, the total amount of steps searching the subtrees is less than the amount of steps the 1 process does, the communications between processes determine the speed.

We also notice that for the same number of processes, using more nodes is slower than just using more cores. So, we can pressume that the communications between nodes are more expensive than between cores.

We could improve the speed of distributed k-NN search, if we chose more carefully the Vantage Point (it is done randomly in this program) , to increase the chance the nearest neighbours will be in the least amount of subtrees and minimize the number of communications.

Finally, it’s worth noting that all the data that might be sent to another process, were stored sequentially in memory in an 1D array, even multidimentional points or 2 different related sets of data(pe. points and their distance from VP/query), so that they would be sent in one message and so reduce the total message overhead in the program.

(https://github.com/KonstantinosChatziantoniou/VantagePointTree-Knn-Mpi)