ARISTOTLE UNIVERSITY OF THESSALONIKI

Department of Electrical & Computer Engineering
HY3606 - Parallel and Distributed Systems

k-NN Search using Vantage-Point Trees

Prepared by: Nikolaos Andriotis 9472
Repository: https://github.com/andriotis/k-NN
Instructors: Pitsianis Nikolaos, Dimitrios Floros
Date: October 15, 2022

1 Implementation

1.1 Implement the vantage-point tree in C

For the purposes of building the tree, I created a structure for the node. Each node, stores the index of the vantage-point selected (in my implementation I always select the last one), the median of the distances of the points to the selected vantage-point and two children nodes. These children nodes, will each recursively build a sub tree with the points distributed to them.

Having this structure, I constructed a function to make the tree, which first computes the distances to the vantage point sequentially and then finds the median of those distances. The median is found by implementing quick-select as the exercise suggests. Then using if statements, I split the points in the inner or outer sets. My implementation corresponds to the first algorithm proposed by Mr. Yianilos.

One important tweak I made, is that the partition function swaps both the distance and the point (they are pointers, not actual information). This, allowed me to create the inner and outer set used by the node's children, solely using bounds, and not if statements (if the current point's distance is less than the median, put it in the inner set).

Another tweak I made, is that instead of just returning after there is only one element in the set, I added another if statement which reduced how many times the make function is called by 40%.

1.2 Parallelize your implementation on multi-core CPUs

To parallelize the making of the tree, I did two things.

Firstly, I parallelized the way I find the distances. This was done, by keeping track of the active threads (the threads that finished their previous assignment and are now available for a new assignment) and equally distributing the set's points among them.

During my experiments, I figured out that my distribution was not equal and that the last thread took the remaining points which bottle necked the other threads. A major improvement to this was to equally distribute the points, but when that wasn't possible, the remaining points would also be distributed equally.

Secondly, I parallelized the way I execute the making of the sub trees. Essentially, if there was at least one active thread, its assignment was to create the inner child, whilst the calling thread would continue with creating the outer child.

1.3 Threshold the parallel calls

In order to threshold the parallel calls, I did three things.

Firstly, I used the real threads of my system as an upper limit. Doing so, allowed me to never create virtual threads, since that would slow down my computations.

Secondly, I set minimum work per thread when calculating the distances. This number was a result of experimentation and I can say that both the number of points and their dimensionality affected its resulting value.

Thirdly, if any of those requirements weren't met, I proceeded to calculate the distances or the inner child sequentially.

1.4 Calculate the all-k-NN

After having built the tree, I use a linked list (using the distance between the query and the node's vantage point as priority), to store my nearest neighbors. The list stores the indices of the closest points, and if it is full, the point with the largest distance, has priority to be removed. If the point to be added, is closer to the query the latter point is removed. The algorithm used for the searching corresponds to the last algorithm proposed by Mr. Yianilos.

2 Results and Remarks

I tested my results by creating a brute force algorithm, using it as ground truth. Seeing the results, both with the sequential and the parallel tree building implementations, I managed to have a significant decrease in execution time from the brute force method. As the dimensionality of the problem increases, we can see that the brute force method falls short, but the other implementations don't, which is actually the reason we build a vantage-point tree in the first place. After the 1,000,000 point mark, my implementations seem to be having a problem. This can be (and will be experimented after the due date of the assignment) a poor minimum work per thread choice of my behalf in combination with my slow system.

Parameters		Brute	Sequential	Parallel
Dimensions	Neighbors			
5	10	0.000268	0.000043	0.000045
	100	0.000402	0.000031	0.000032
	1000	0.027510	0.000027	0.000027
50	10	0.001944	0.002510	0.001921
	100	0.002371	0.002155	0.002161
	1000	0.034177	0.002168	0.002150
500	10	0.007679	0.009211	0.009264
	100	0.009520	0.009144	0.009620
	1000	0.036159	0.009855	0.010172

Table 1: 10,000 points, measured in seconds.

Parameters		Brute	Sequential	Parallel
Dimensions	Neighbors			
5	10	0.001307	0.000040	0.000122
	100	0.001563	0.000032	0.000045
	1000	0.054113	0.000033	0.000039
50	10	0.009969	0.000041	0.022734
	100	0.009181	0.000032	0.022800
	1000	0.059874	0.000032	0.023037
500	10	0.076945	0.000042	0.094678
	100	0.077096	0.000032	0.119798
	1000	0.129256	0.000033	0.109970

Table 2: 100,000 points, measured in seconds.

Parameters		Brute	Sequential	Parallel
Dimensions	Neighbors			
5	10	0.013352	0.000114	0.000109
	100	0.013585	0.000042	0.000032
	1000	0.087254	0.000038	0.000031
50	10	0.091335	0.278538	0.261932
	100	0.091609	0.297470	0.261933
	1000	0.170804	0.304817	0.261710
500	10	0.829947	1.394901	1.163626
	100	0.837127	1.336409	1.136240
	1000	0.922903	1.314896	1.222550

Table 3: 1,000,000 points, measured in seconds.