Report KDtree

Zainab Nazari

January 2022

1 Introduction

In this note we study the KDtree. KDtree is space partitioning algorithm for a set of data in D dimensions. We build a two dimensional search tree

2 Algorithm and implementation

The algorithm is implemented as a recursive function. It takes a range of K dimensional points defined using a pair of iterators and an integer. The latter is interpreted as an axis, which is used to determine how we divide the set of points.

During every iteration, we advance current axis number to the next one. We create a new node. The axis number is stored in the node and it's also used for sorting the data set. We sort the points by their coordinates related to the axis using quicksort algorithm. We also assume that the points are distributed homogeneously. With the assumption, it is possible to divide the sorted points just by getting median value from them. We store the median as the current node's value.

The function returns a pointer to a node. So, after all that we recursively call the same function for the left and right subsets to get left and right child nodes. At the end we return the current node.

We must not forget the base case: if the number of points during any iteration is 0 - we return null.

We used OpenMP for multithreading. We used task constructs so that we could recursively work with multiple threads in parallel.

3 Performance Analysis

There two strategies at each step we should divide in the median point, in order to find the median it takes a number of operation to sort them.

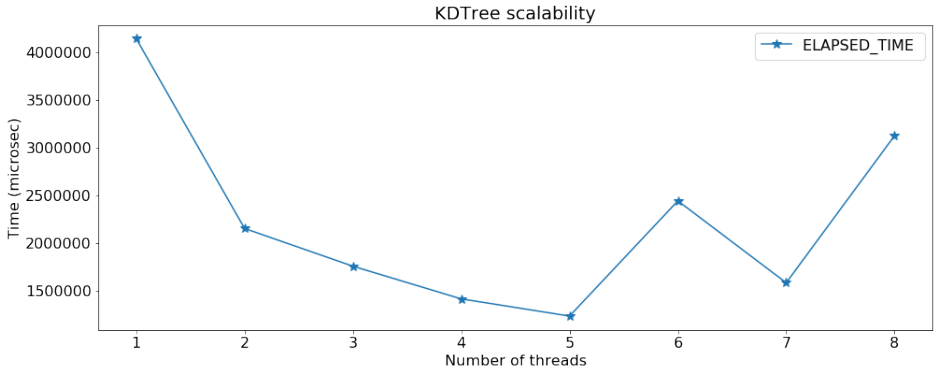
The other option is to estimate the median the point that is closest to the middle of the smallest and biggest value the advantage is easier to compute. the disadvantage is that since it is not median the tree gets very imbalanced. Sine

the data set that we have to generate has outliers. Then the median would be very imbalanced. But since we consider that the data point is homogeneous the outlier data point will not ruin the strategy badly.

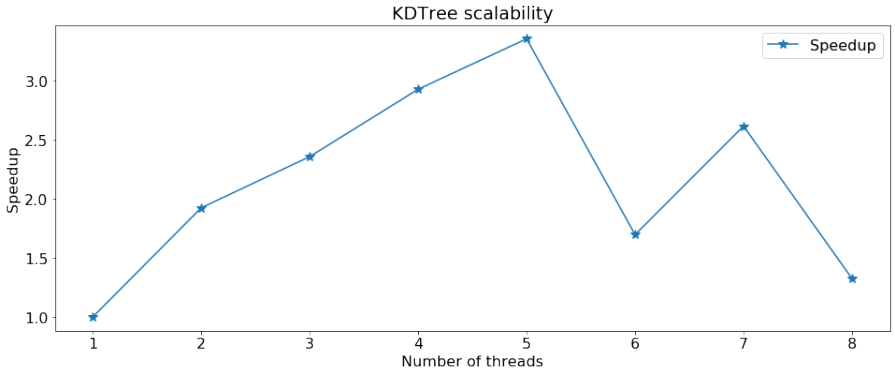
The strategy with estimating the median is more convenient even though the operations are more to be conducted. For the median for each layer to sort the data we need to do `n*log(n)` order of magnitude operations per each direction once. Whereas for the first strategy we need `n` order of magnitude operations per each layer.

4 Performance model and scaling

For openMP we study the strong and weak scalability. For the strong scalability we set the number of data points and increase the number of threads and measure the time and speed up. For the weak scalability we increase the number of data points proportional to the number of theads and measure the time and speed up. The weak scalability is also done, and the job scripts is given, also the data. However, we do not see a smooth weak scalability, there are many imbalanced result.



and the plot for the speedup:


KDTree scalability
Speedup
Speedup
3.0
2.5
2.0
1.5
1.0
1
2
3
4
5
6
7
8
Number of threads