

Name of student: Chieh-En Li

Name of instructor: Milind Kulkarni

Project title: Approximate K Nearest Neighbor with KDtree on GPU

Credit hours: 1

Term: Spring 2020

Brief description of the project:

In this project, I am looking for any method to optimize K nearest neighbor search with KDtree running on GPU and try to lower the time consumption without sacrifice too much accuracy.

Brief summary of the results:

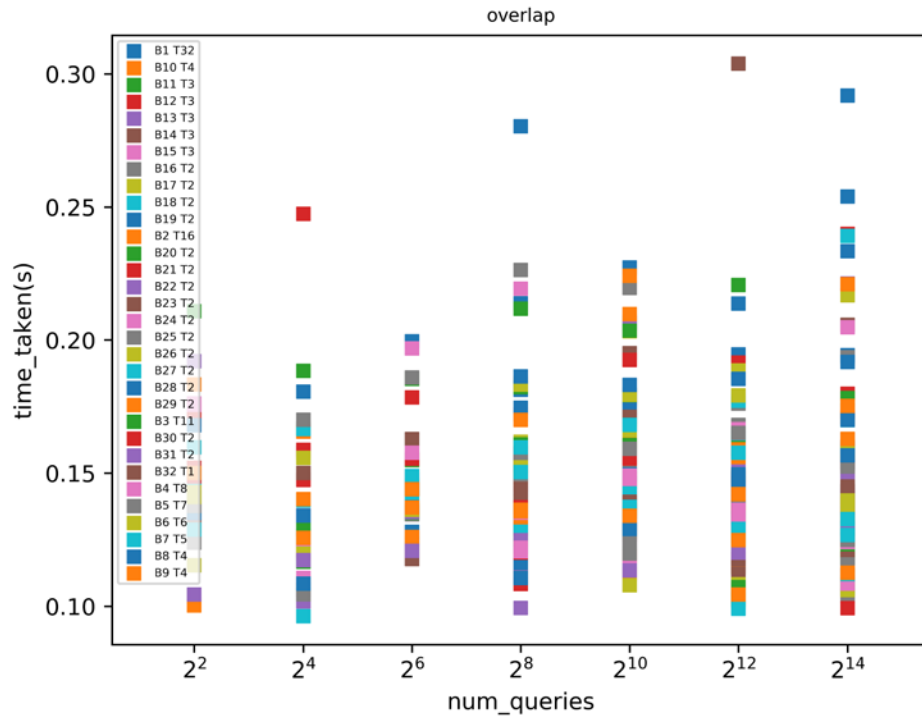
After my experiment, I found just purely control the the traverse way may not increase any time efficiency but may get a worse result. Therefore, to solve the problem may modify the original KDtree, especially decrease the tree depth, but decrease the tree depth may sacrifice the accuracy.

Outline for the report:

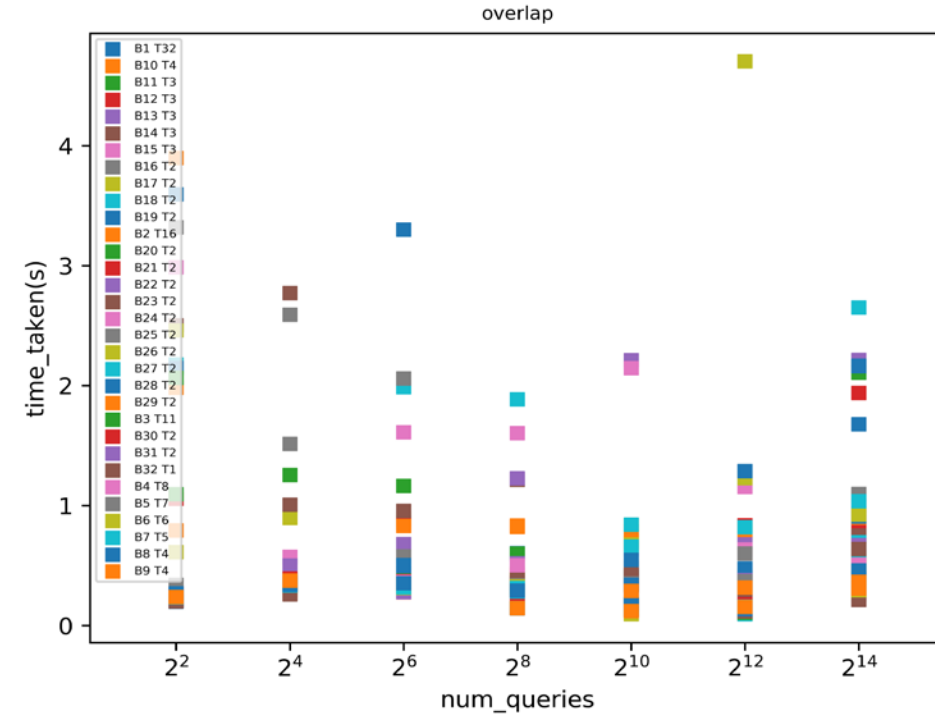
1. Result comparison and possible reason.
2. Method.
3. What I have learned from this experience.

Result Comparison and Possible Reason.

Non –reversed tree order result



Reversed tree order result



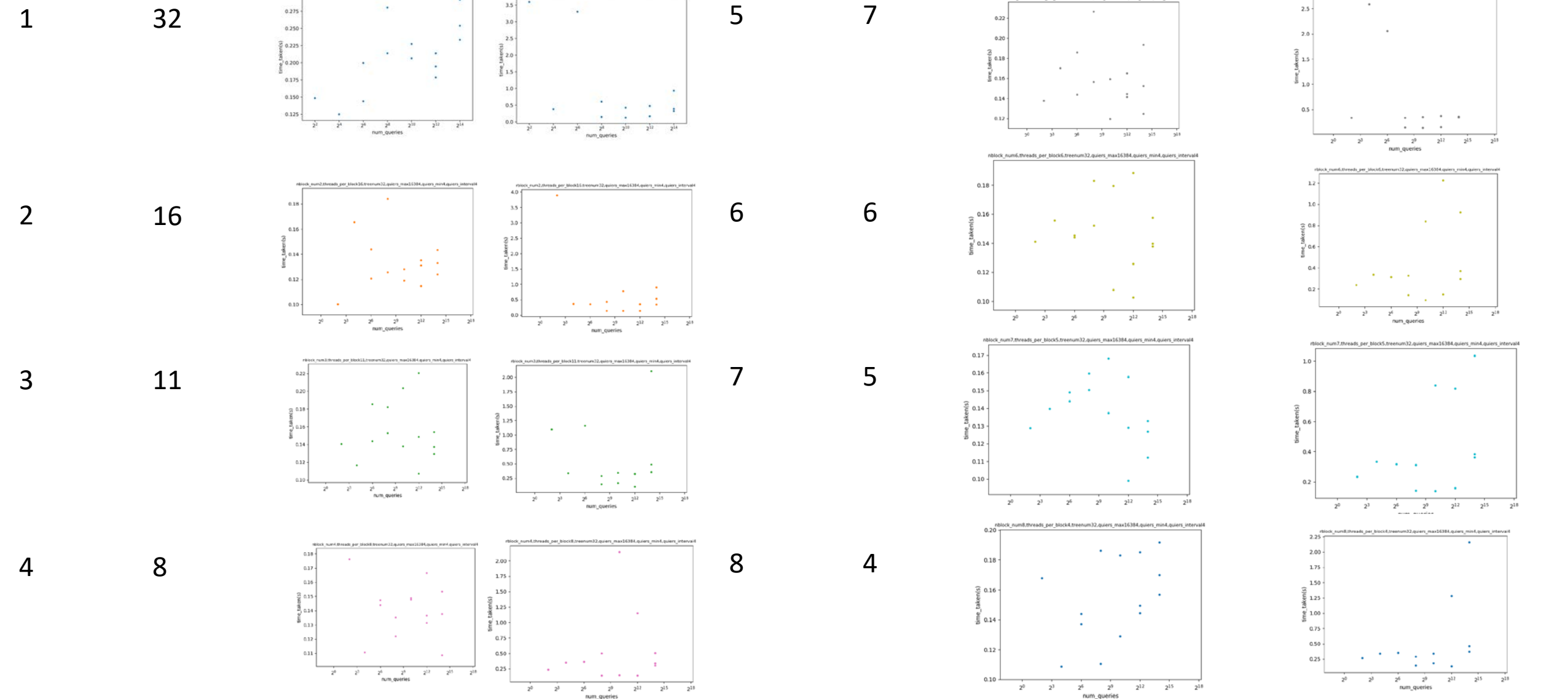
In the legend “BxTy”, x means block number and y means each thread number in each block.

The experiment condition: same queries points and on the same machine.

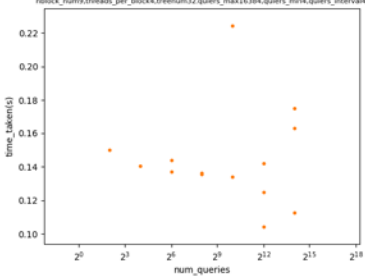
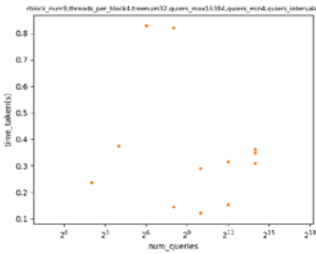
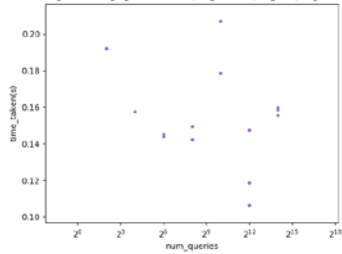
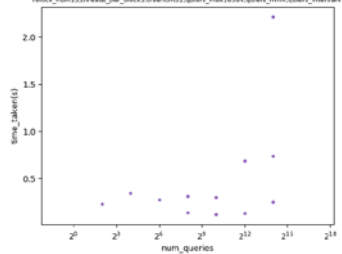
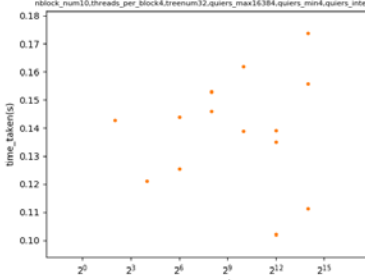
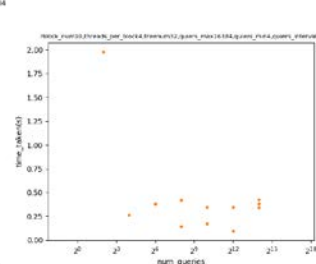
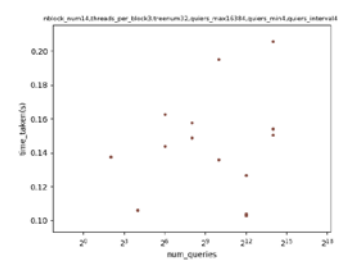
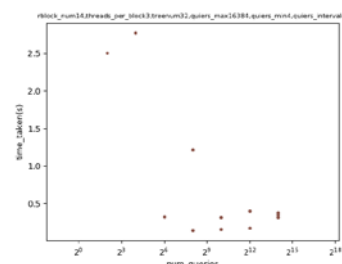
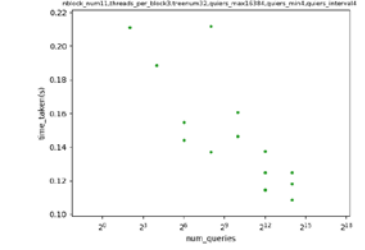
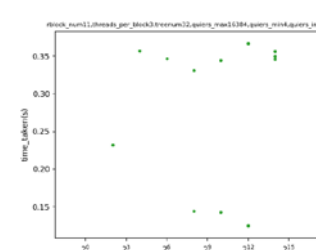
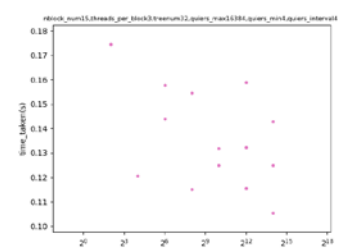
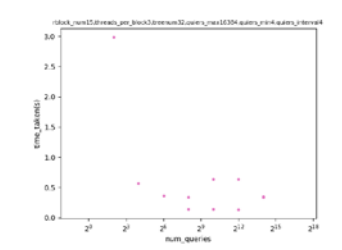
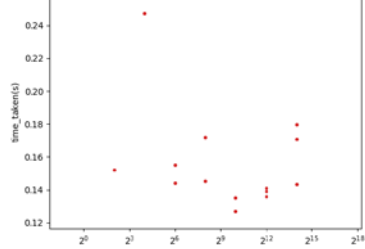
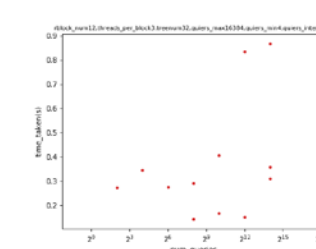
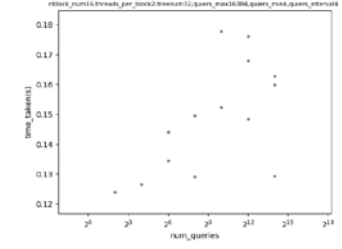
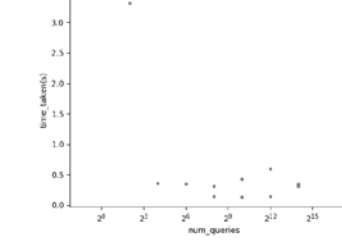
As the result, reversed tree order can get an overall lower time efficiency; there is no linear relationship between time for traversing and block number. The reason that reversed tree order can lead to a relatively longer time on traversing could be the unideal spatial locality. When the machine traverse the reversed order trees, it cannot access just the just the adjacent memories blocks.

Result Comparison and Possible Reason.

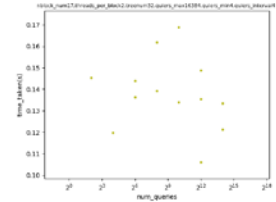
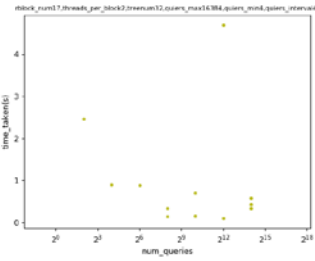
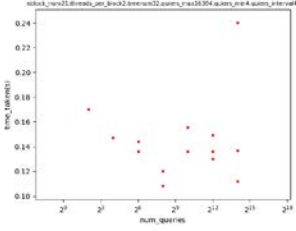
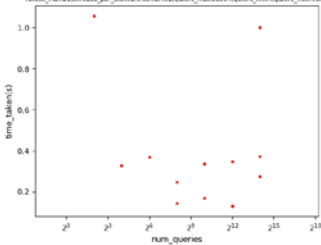
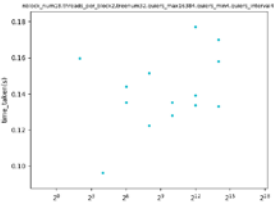
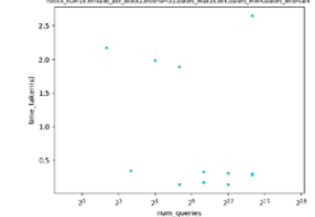
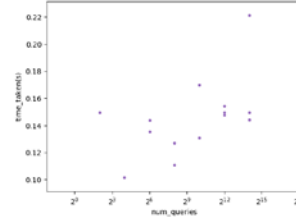
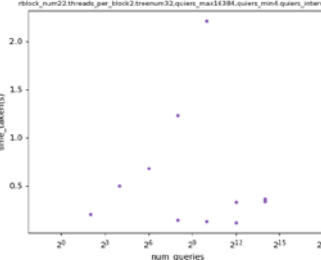
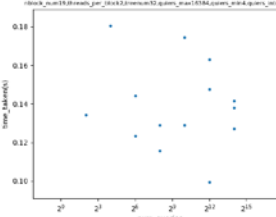
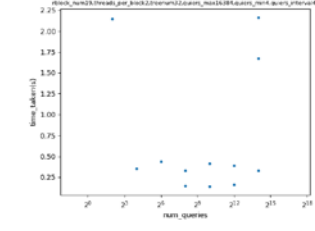
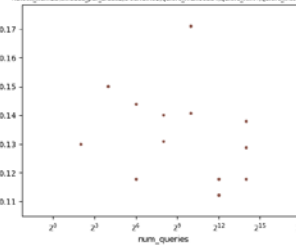
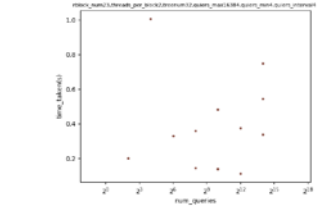
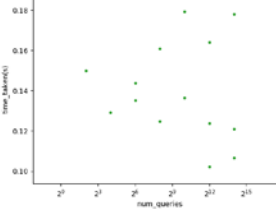
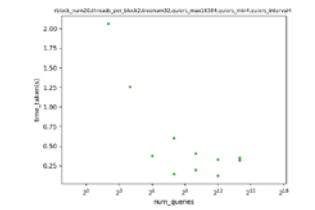
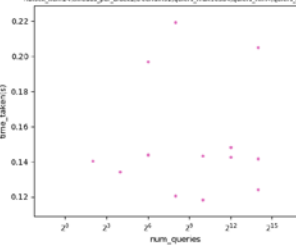
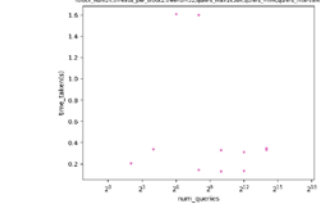
Block number.	Thread number for each block.	Non –reversed tree order result	Reversed tree order result	Block number.	Thread number for each block.	Non –reversed tree order result	Reversed tree order result
---------------	-------------------------------	---------------------------------	----------------------------	---------------	-------------------------------	---------------------------------	----------------------------



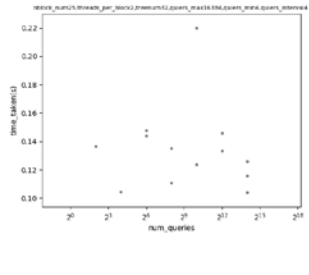
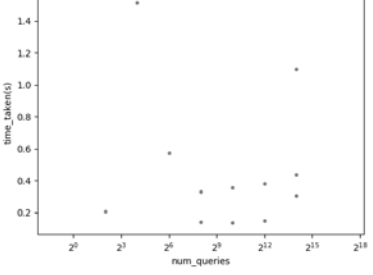
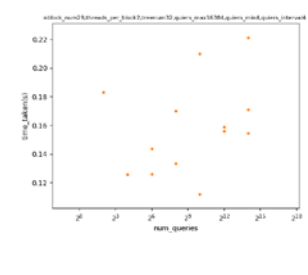
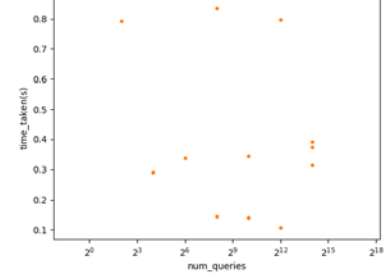
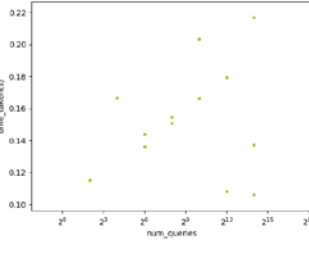
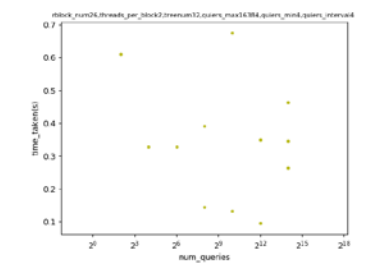
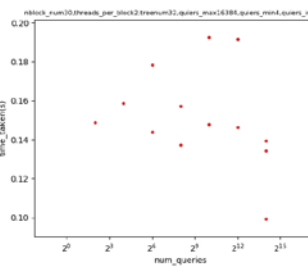
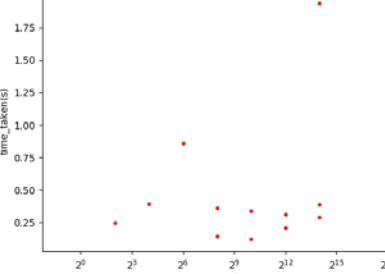
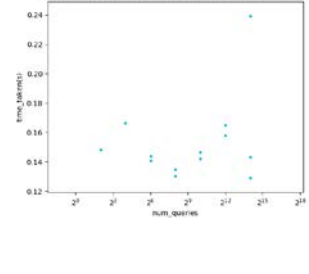
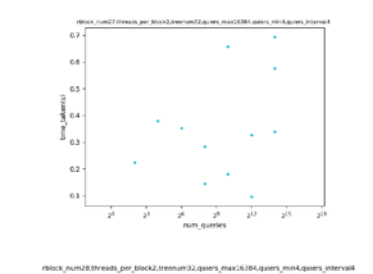
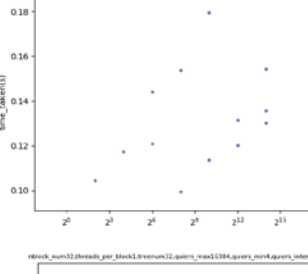
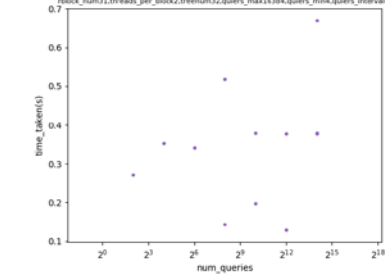
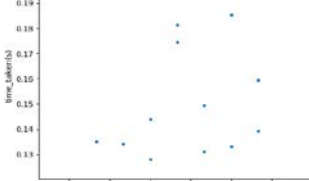
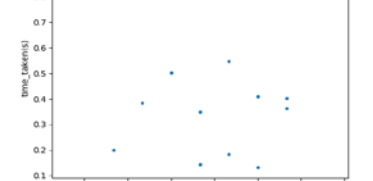
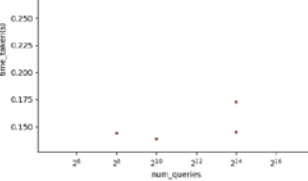
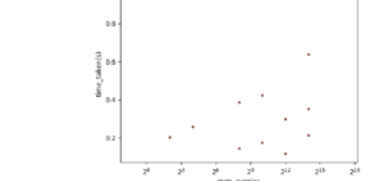
Result Comparison and Possible Reason.

Block number.	Thread number for each block.	Non –reversed tree order result	Reversed tree order result	Block number.	Thread number for each block.	Non –reversed tree order result	Reversed tree order result
9	4			13	3		
10	4			14	3		
11	3			15	3		
12	3			16	2		

Result Comparison and Possible Reason.

Block number.	Thread number for each block.	Non –reversed tree order result	Reversed tree order result	Block number.	Thread number for each block.	Non –reversed tree order result	Reversed tree order result
17	2			21	2		
18	2			22	2		
19	2			23	2		
20	2			24	2		

Result Comparison and Possible Reason.

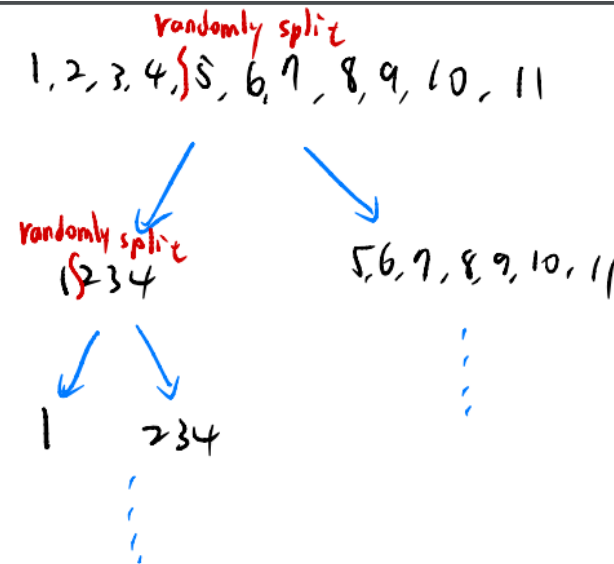
Block number.	Thread number for each block.	Non –reversed tree order result	Reversed tree order result	Block number.	Thread number for each block.	Non –reversed tree order result	Reversed tree order result
25	2			29	2		
26	2			30	2		
27	2			31	2		
28	2			32	1		

Result comparison and possible reason. Conclusion.

- Based on above results, there is no obvious best block and thread pairs to get the lowest time consumption. However, when traversing the reversed order tree, only when block number is 9,11,21,23,26,27,28,29,31,32 would get lower time on traversing but still get a higher traverse time than non-reversed tree order.
- To lower the time taken, instead of modifying the block and threads, cut the traverse depth would be a better way to enhance the time efficiency.
- To modify the inquiry point order may not work in my project because the method to accesses the inquiry points is in random when building a KDtree.

Method

- First I crease a KDtree with random splitting an array of inquiry points as the figure shown.



Method

- Second, I compare the inquiry points and the target with each dimension when traversing. When traverse depth increase, the compare dimension increases until reach the maximum dimension and then compare with the first dimension.
- In my experiment, I build KDtrees with three dimensions.
- To optimize the traverse process, I control the block number and threads number in each block as the figure shown bellow.

```
k_nearest_search_k1_GPU << < block_num, thread_num_per_block >> > (tree_reversed, target, tree_num, found);
```

What I have learned from this experience.

- Kdtree build and traverse.
- Cuda.
- Properly statistic showing.
 - A 3D graph is not intuitive for showing the concept.
 - Sometimes matplotlib may not capture the correct order of a dataset, so scattering the points may be better.
 - If too many data points may not clearly see the result because the noise would be enlarged as well.

