Exercise IV

Parallel & Distributed Systems

An implementation of Vantage Point Tree in CUDA

Evripidis Chondromatidis 8527 eurichon1996@gmail.com

Github Code: https://github.com/eurichon/CUDA_PROJECT_4.git

Problem & Aim

A vantage point tree is a structure which can efficiently partition the data in the n-dimensional space. In this project we aim to parallelize the process of creating one such structure by utilizing the capabilities of GPU multithreading using CUDA. In order to succeed high performance, we aim to improve the following things which will discussed below:

- Parallelize the calculation of distances between the subtree points and the vantage point which is currently selected.
- Parallelize as much as possible the process of finding the Median radius in each subtree.
- Avoid as much data transferring between the Host(CPU) and the device(GPU) which has a very high cost.
- And finally, create each tree level in parallel, resulting in a iterative process.

Distance Calculation

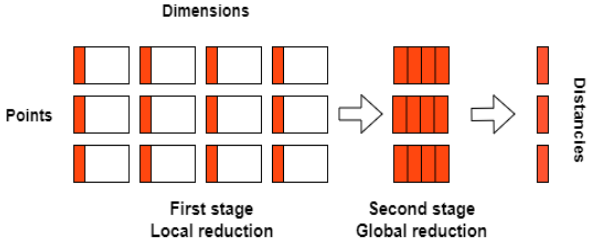
In order to calculate the distances efficiently we vectorize the process using the formula:

$$D(X,Y) = XX' - 2XY' + YY'$$

The implementation consists of:

- a function which finds the best block and grid dimensions according to the given number of point and their dimensions which aims to assign one partial product per thread, while keeping the dimensions of the blocks multiple of 32 and having as few wasted threads as possible.
- A stage where the vectors are divided and copied intro block of shared memory where the partial products are calculated, using the above formula. This stages also reduce the products locally and saves them in an in-between array.
- A second and final stage which uses the in-between array to globally reduces the product to the final result distance. At this stage it is guaranteed that the size y of the in-between array can fit in one block at the y direction.

This whole process allows to be processed points with arbitrary number of dimensions which may exceed the shared memory of a single block.



Median Calculation

During the construction, in order to partition each subtree, we have to find the median distance which split the data into two halves. This is usually be done with Quickselect. However, **Quickselect** has a very low degree of parallelization as it is not even belonging to the divide and conquer algorithms category. This is because only the one branch during each recursion is developed. Instead we will use a custom implementation of the famous **Bitonic sort**, which at each stage uses a constant amount of threads (half the size of the data) and each runs independently.

After a lot of trial and error I managed to find the formula that maps each thread to the write pair of values and with the correct comparison direction depending on the iteration level.

$$At\ a\ given\ step\ of\ comparison\ let:\quad a = Thread_{id} div\ step\ \ \&\ \ b = Thread_{id} mod\ step$$

$$Then\ \ dir = (a\ \%\ 2 = !\,dir)\,, pos = a * (step \ll 1) + b, comparing\ positions\ (pos, pos + step)$$

This formula is the base of my implementation, with a slight change to the direction calculation which has to remain constant inside the inner loop (merging) and equal to the value it had on the outside loop(sorting). Below is an example for 8 randomly selected elements belonging [1, 8].

At iteration with step=2 and dir=1 (ASCENDING)

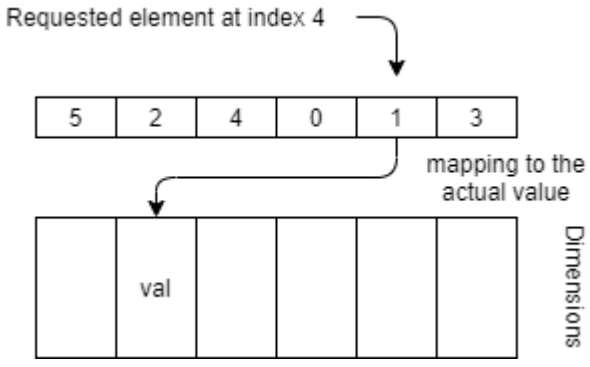
Thread Id's	a	b	pos	pos + step	direction
0	0	0	0	2	1
1	0	1	1	3	1
2	1	0	4	6	0
3	1	1	5	7	0

pairs of values to be compared

Reducing Data Transferring

While GPU offers a great amount of parallelization its capabilities is often limited from the data transfer bandwidth between the Host and the Device. Therefore, in order to make our implementation as efficient as possible we will have to limit all unnecessary data transfers. Following this logic, we aim to create the whole this inside the device (GPU) memory.

In addition, the process of storing all the dataset for each tree level in the device memory is very costly as well as every possible swapping of points, especially in the high dimensional space. So, we chose to simulate the pointer, using an index map which is stored instead of the actual points and which maps the request elements to the actual elements. This allows us to perform the index swaps alongside the Bitonic sorting. With that in mind we overload the original Distance calculator and Bitonic sorting to access the elements throughout an index map.



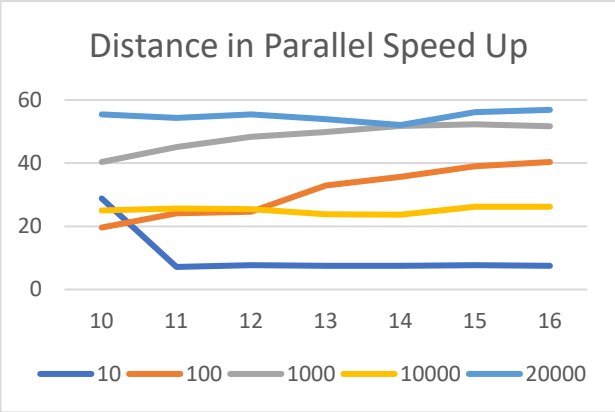
Construction of Tree level in Parallel

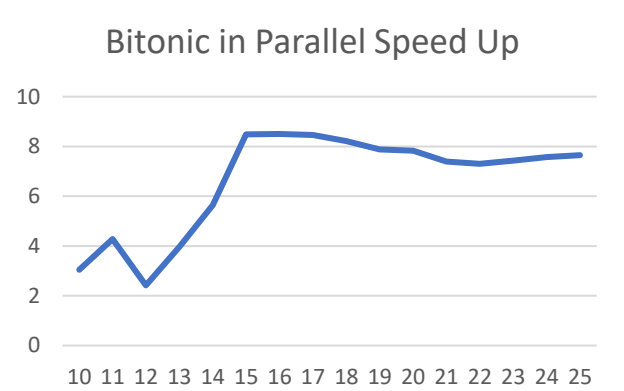
To accomplish this we have to pass the number k of subtrees that current level should have in the overloaded function. And now by performing module operation of the thread_id with the (n/2*k) we

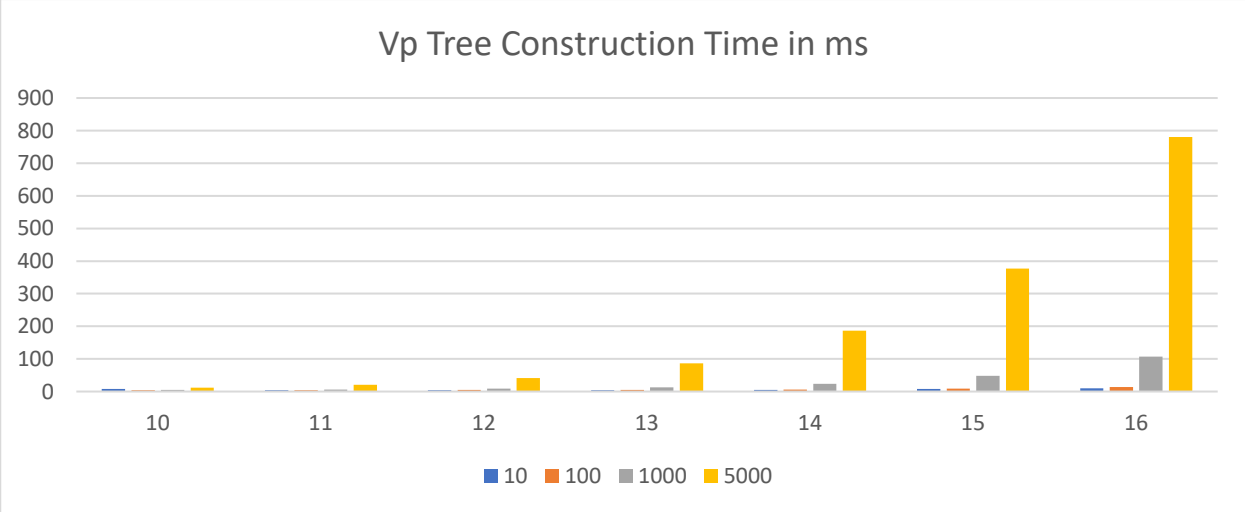
will force the thread to redo the same operation in their respective range. To guide the threads to the correct data we have to add an offset equal to: (thread_id / (n/(2*k)) * (n/(2*k).

This will result in this tree structure, where each level store the produced indexes, and the first point always of the each subtree is selected as vantage point. The we sort each subtree in descended order to push this element to the back and reshuffle the v-point.

Results in logarithmic scale







Comment

Currently the code has some bugs, here is presented the idea where the first goals are meet the tree isn't yet functional. The distances aren't stored. Also in its current design it will work for input of power of two. The other parts of the assignment weren't in a proper manner so they are not include.

Machine Specifications:

CPU: Intel Core I7-3300HQ **GPU**: GeForce GTX 940 Ti **RAM**: 8GB **OS:** UBUNTU 20.04