Processing Query Points for kNN in Batches on CUDA

prepared by Andrew Gan

Previous method

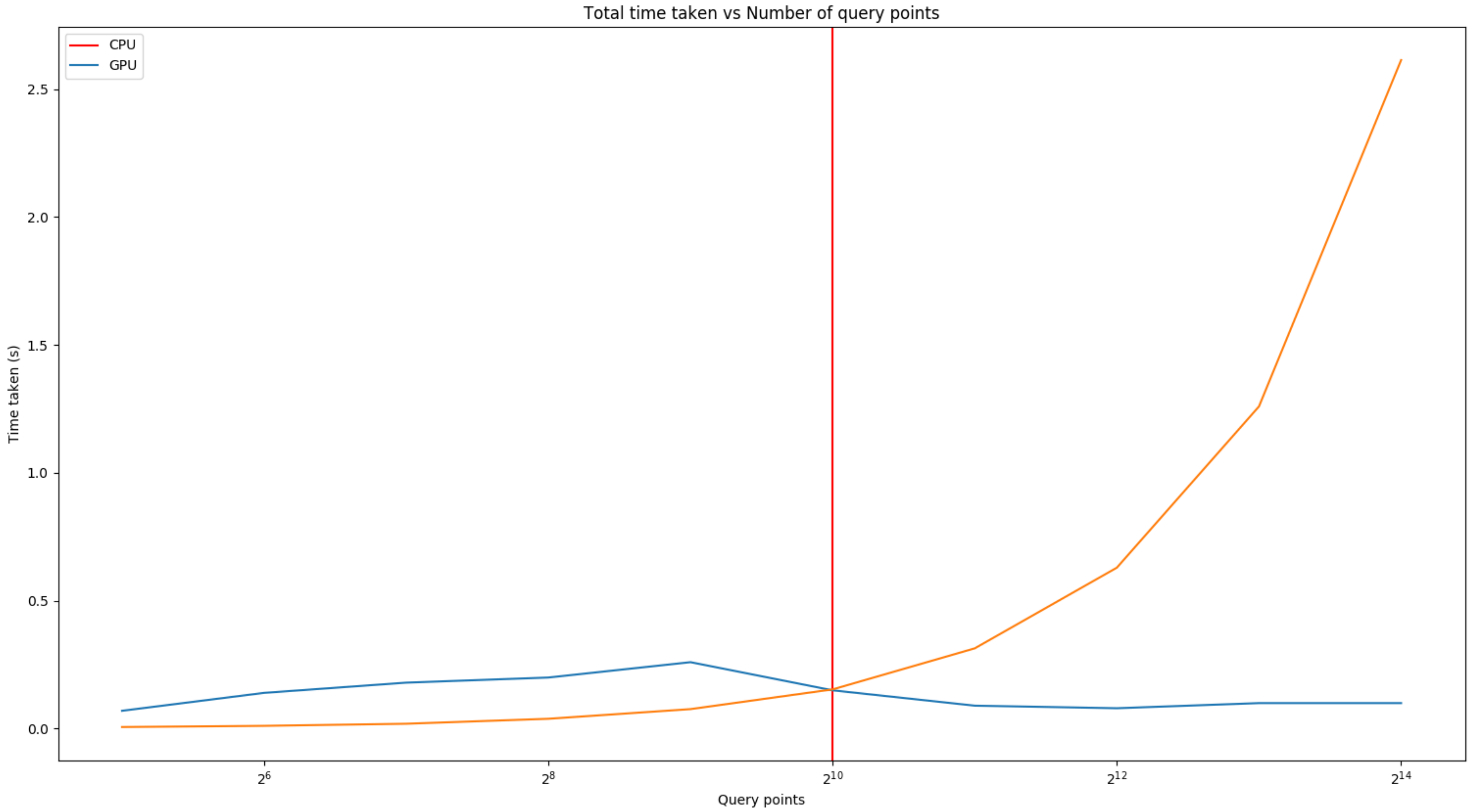
- Receive a collection of query points for kNN and allow them to traverse a hybrid (spill + metric) tree from root to leaf node

- Number of threads and blocks allocated follow the basic principle:

- Given number of queries q, find number of blocks b,

  - if q <= 512: b = 1, block has 512 threads

  - if q > 512: b = [(q + 511) / 512], all blocks have 512 threads

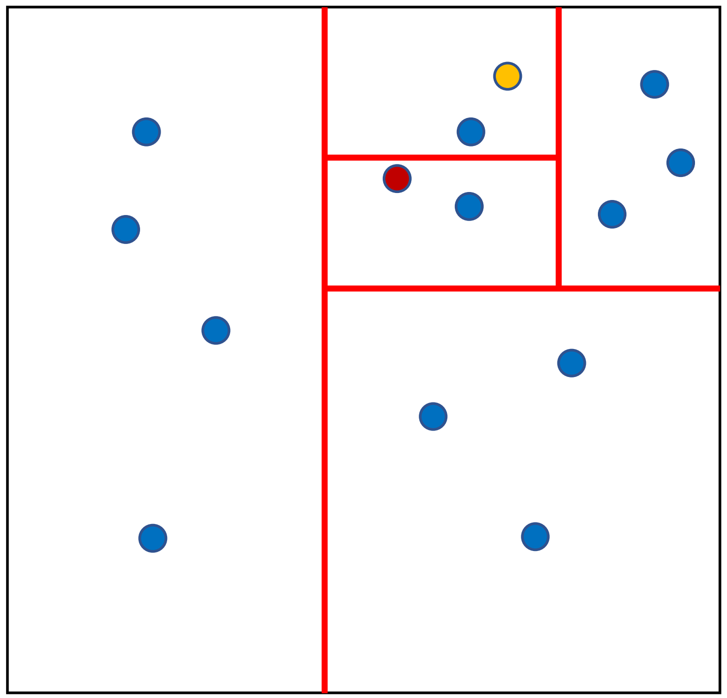

Total time taken vs Number of query points
CPU
GPU
2.5
2.0
1.5
1.0
0.5
0.0
Time taken (s)
2^6
2^8
2^{10}
2^{12}
2^{14}
Query points

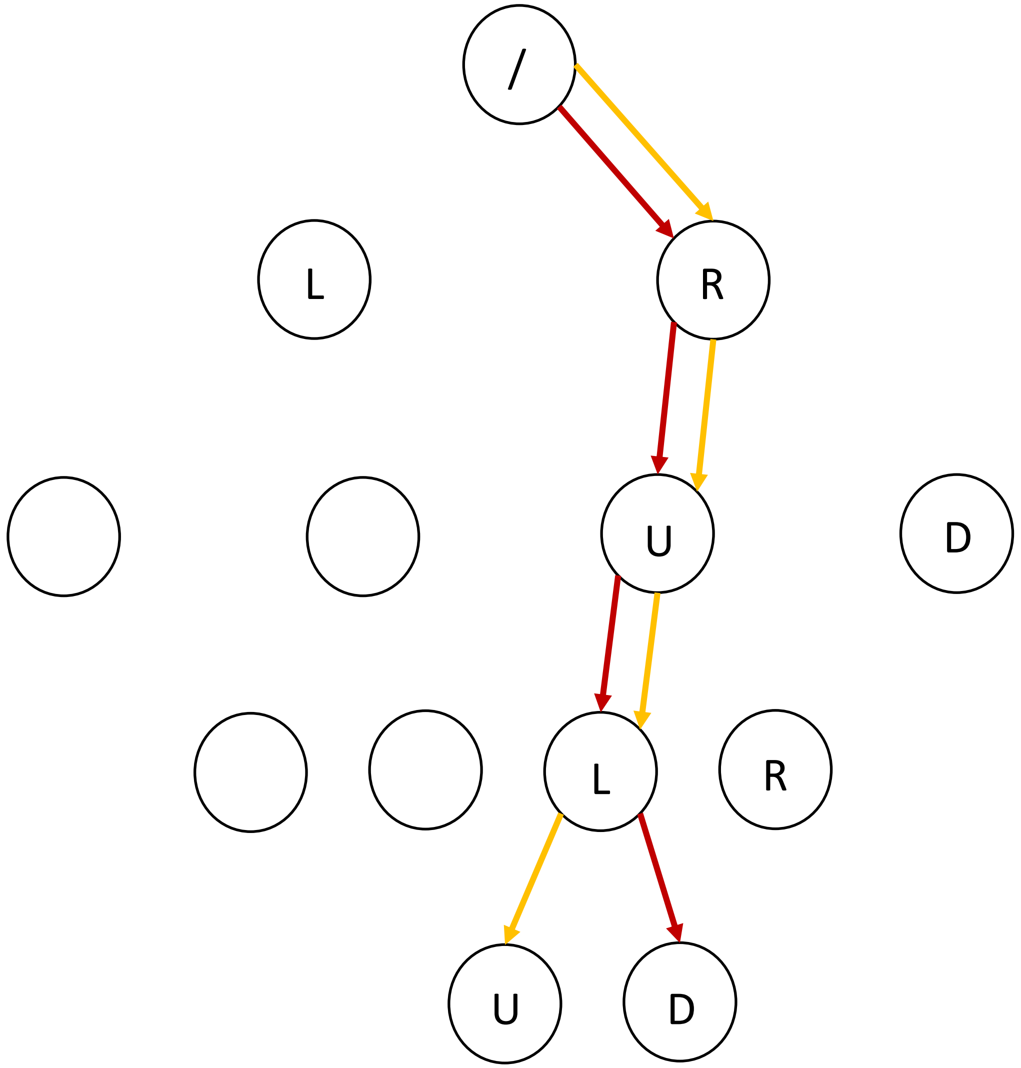
Observation

- Notice that query points that are nearby may traverse through the same first few nodes from the top of the tree
- This becomes more apparent when the query points are more densely located rather than evenly distributed
- There must be a way to exploit the locality of query points that are close together so that repeated paths can be explored in one iteration

● ● query
● data

The two points near each other have explored the same path for four layers before diverging at the fifth layer



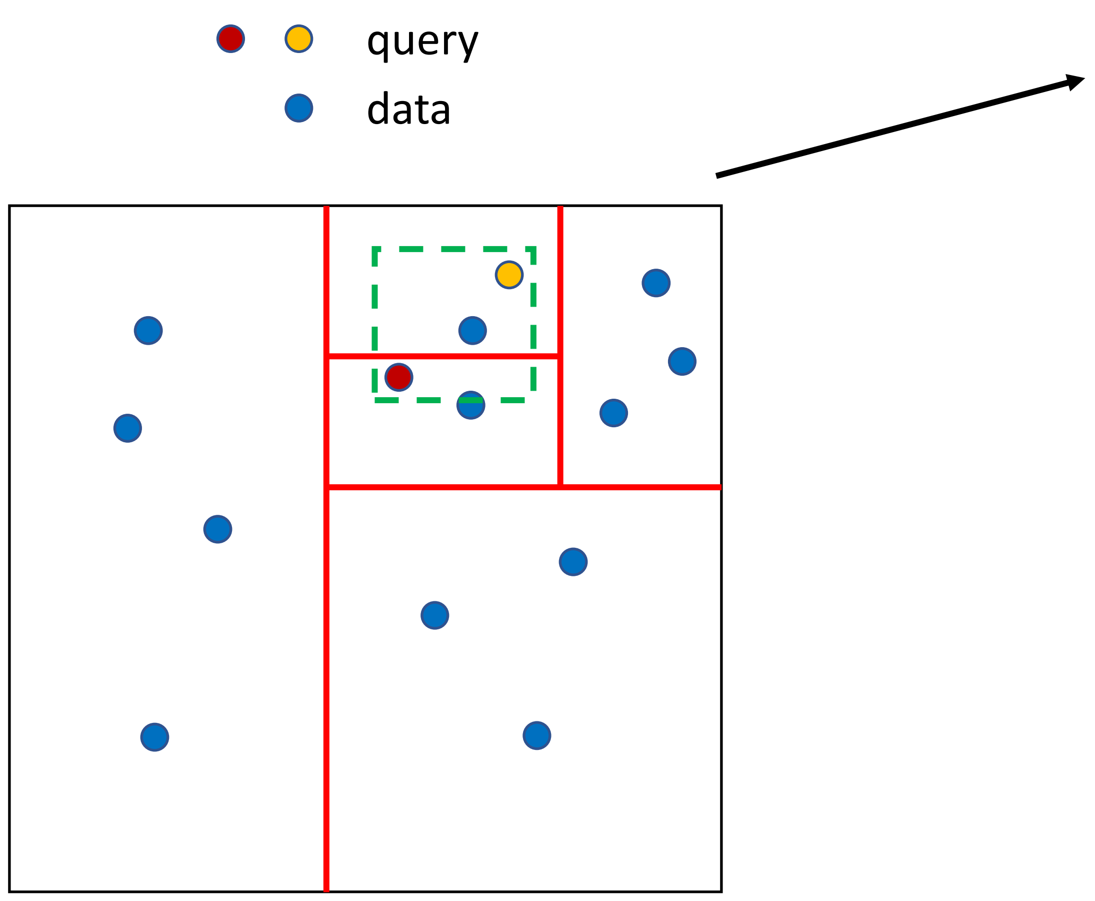


Solution: Batch Processing of Queries

- New structure: GroupPoint

```
struct GroupPoint {
    int continueNode;           // node in tree at which to begin diverging the query points
    Point startPoint;           // min position encompassing the group of query points
    Point* endPoint;            // max position encompassing the collection of query points
    vector<int> queryIds;       // original position of queries in list before being grouped
    Point* memberPoints;        // points that are contained by this group
};
```

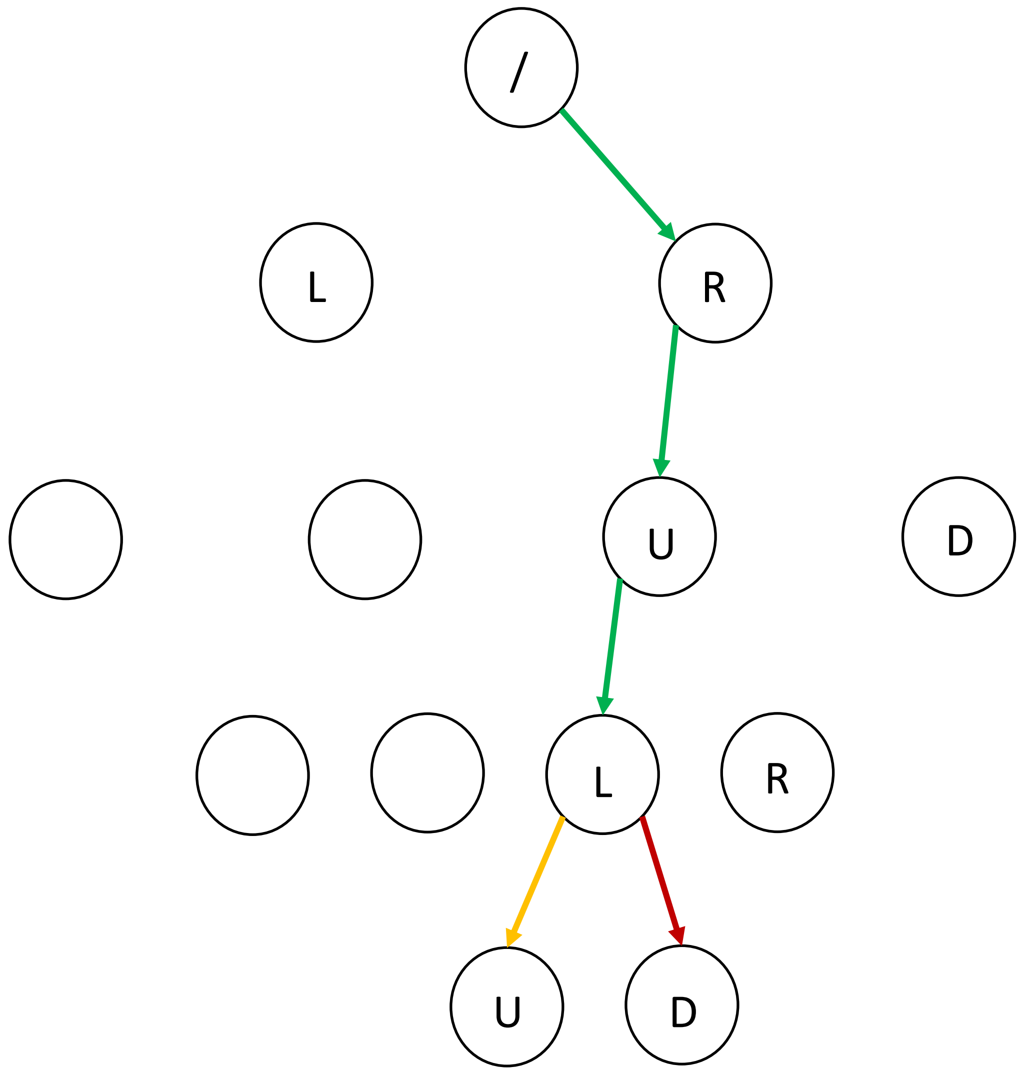
- When a list of query points are received, the query points (assumed to be randomly distributed) are partitioned and placed in groups based on locality



GroupPoint 0

Member points: ● ●

Boundary: depends on member points



GroupPoint is treated as a single entity and used to traverse the tree until either:

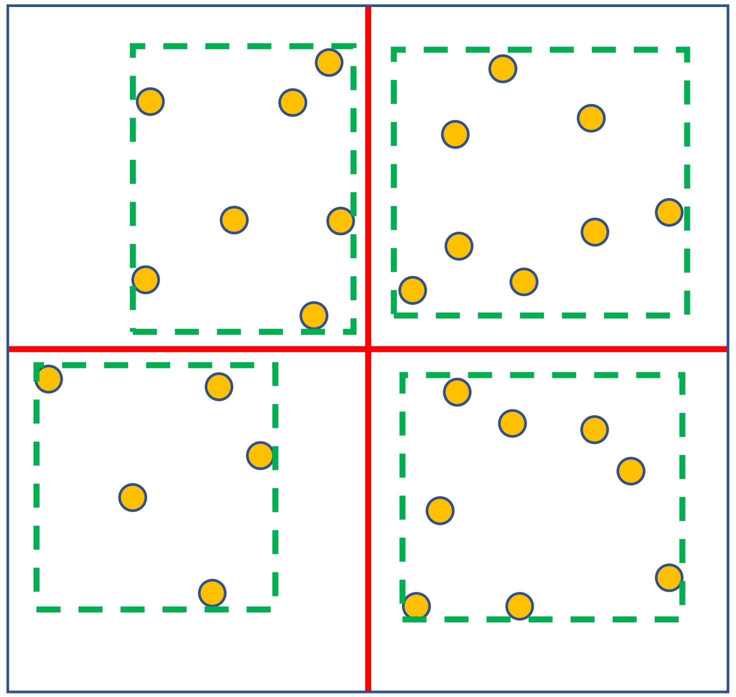
i. Allowed number of layers to traverse is reached
ii. Neither child nodes fully contain the GroupPoint

How are query points grouped?

- For early testing, it is assumed that query points are distributed evenly enough such that grouping them by their position will suffice

- Work in progress

- K-means clustering might work?



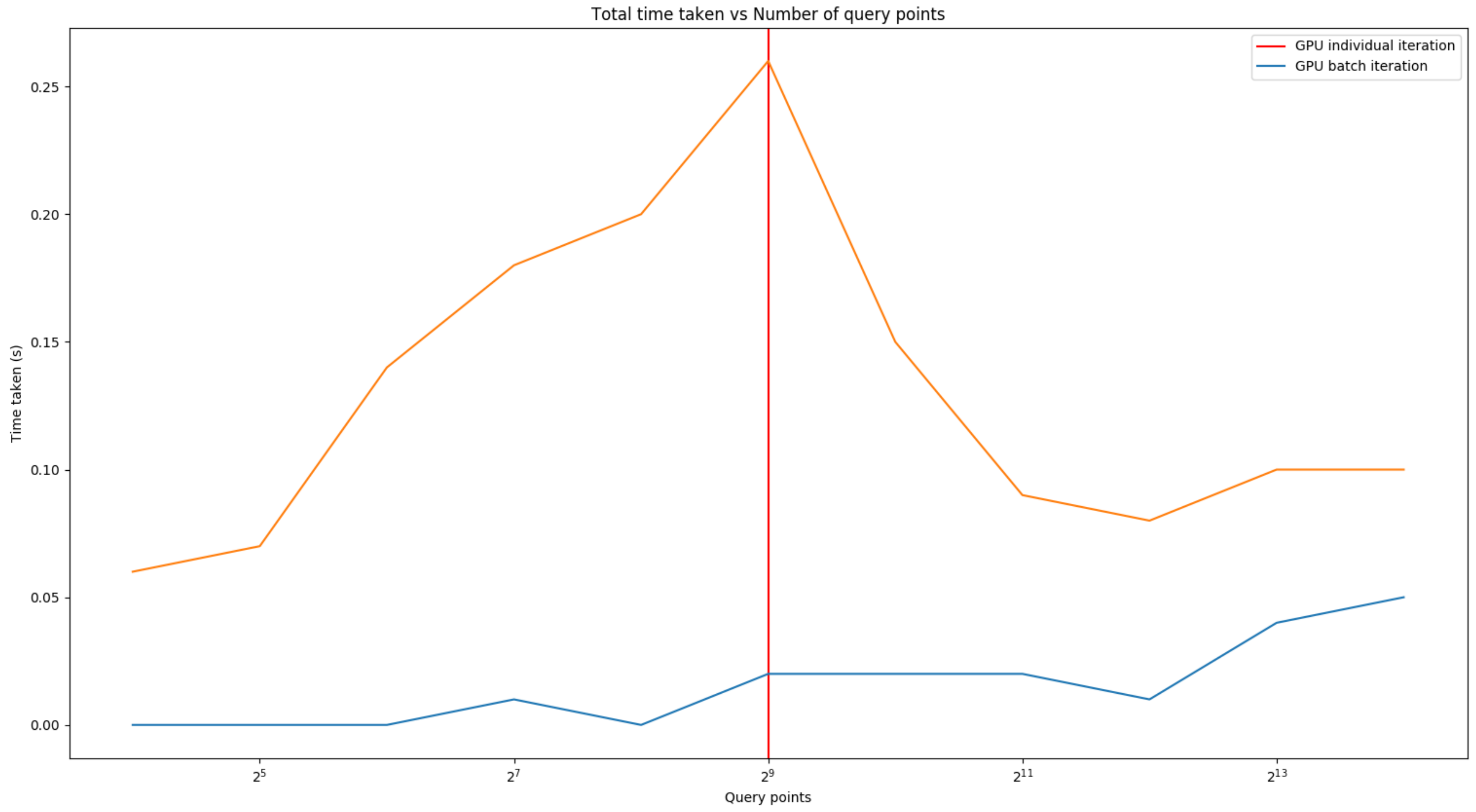
Actual implementation

- Every GroupPoint will traverse the tree on the CPU side until the node where the GroupPoint must disperse its member queries.
- We’ll call the node where this happens the ‘continueNode’
- All query points belonging to the same group should have the same ‘continueNode’
- The queries, along with their ‘continueNode’ are now cast into a flat array and passed into the device function
- Instead of always starting the traversal from node 0 (root node), the device function allows the traversal to begin at the corresponding ‘continueNode’ of the query being processed.

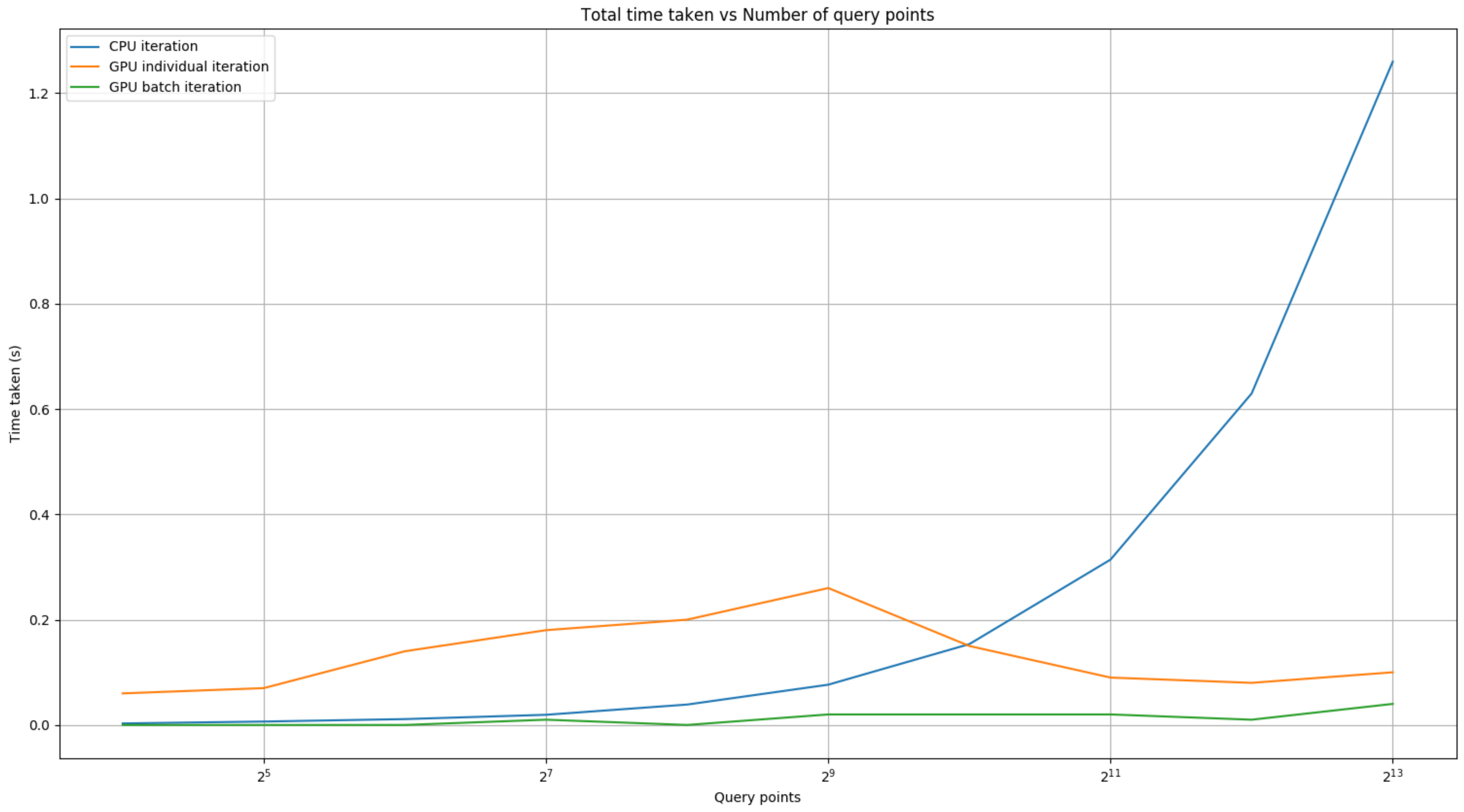
Preliminary Results

- Speed up of the traversal depends on how many layers of the tree is the GroupPoint allowed to traverse before dispersing
- Results for the very possible reduction in accuracy coming soon…
- The spike for individual iteration at 512 query points is likely due to that being the maximum number of threads per block. Once that number is exceeded, more blocks are allocated, and tasks are more efficiently divided
- Cannot explain why the spike is reduced in batch iteration
- More thorough testing needed to verify results

Preliminarv Results



Preliminary Results



Possible improvements in the future

- More reliable method of grouping the query points:
  - K-means may work
- Allowing the GroupPoint to traverse the tree on the GPU side as well
  - An implementation issue because of complex structures
- Tidy up the cuda malloc and free functions so that the process of copying data from CPU to GPU memory is more efficient