Parallelized KD Tree for Parallel k-Nearest Neighbors Search

Background: KD Trees

KD-Tree Basics:
- A space-partitioning data structure for organizing points in a k-dimensional space.
- Ideal for multidimensional key searches, like nearest neighbor and range searches.

Structure and Function:
- Splits space into "left" and "right" at each node using one dimension at a time.
- Alternates between dimensions to maintain a balanced tree, aiding in efficient search.

Key Benefits:
- Facilitates fast search operations by reducing dimensionality one level at a time.
- Adapts to the inherent distribution of the dataset, which can prevent skewed partitions.

Practical Applications: Common in areas requiring fast spatial searches, like computer vision and geographic data systems.

Optimal Use: Most effective in spaces with fewer dimensions due to the curse of dimensionality in high-dimensional spaces.

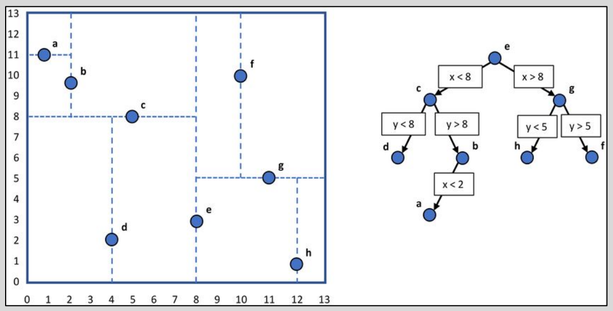


Figure 4: KD-Tree Figure
https://www.researchgate.net/figure/An-example-two-dimensional-k-d-tree-k-2-built-from-nodes-a-through-h-Dividing-planes_fig2_314298746/actions#reference

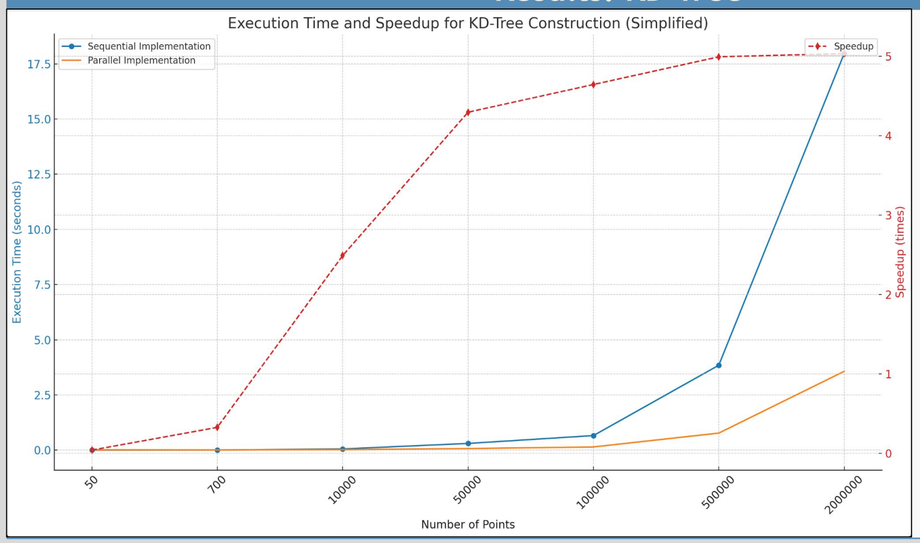
Lock Free KD Tree

Lock-Free KD-Tree Structure:
- Utilizes atomic operations for synchronization rather than traditional locking mechanisms.
- Supports concurrent insertions by multiple threads without the need for mutex locks.

Concurrent Insertions:
- Employs compare-and-swap (CAS) operations to ensure that nodes are inserted correctly even in the presence of concurrent modifications.
- CAS operations allow a thread to proceed with node insertion only if the observed state has not changed since last checked.

Performance and Scalability:
- Significantly reduces contention among threads, leading to performance gains as the number of concurrent operations increases.
- Provides better scalability, as the overhead does not grow with the number of threads, unlike mutex locks that can become bottlenecks.

Results: KD Tree



We see a clear effect of parallelism in the graph.

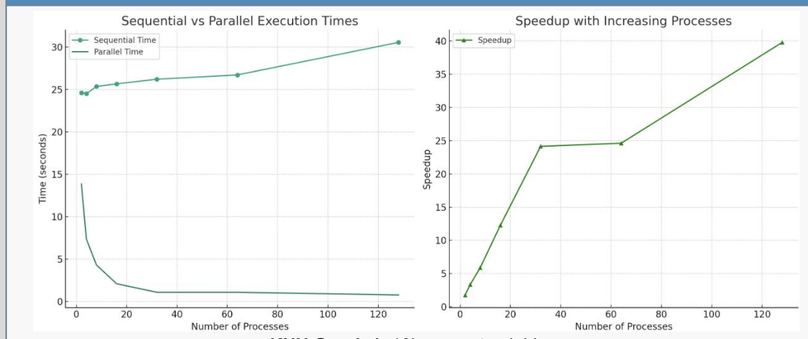
As the number of points becomes 2 Million, the execution time of the sequential code shoots up to more than 15 seconds.

The parallel implementation, however, only went to 2.5 seconds, with a speedup of over 5x for 8 cores (GHC Machines).

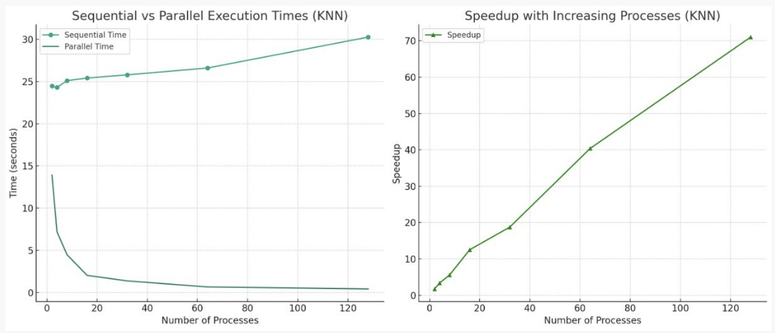
The red line in the graph represents the speedup observed, and it can be seen that the speedup also increases dramatically with the number of points.

The delays caused by execution overhead and task scheduling are compensated well by the benefits of parallelism.

Results: KKN



KNN Graph 1: 10k nearest neighbors



KNN Graph 2: 1k nearest neighbors

Both tests are run on PSC machines

Number of cores range from 2-128 by powers of 2

Target point: {0 1 2 3 4 5 6 7 8 9}
Dataset:
- Number of points: 2 million random points
- Labels: range from 1 through 5
- Number of features: 10

Conclusions:
- Speedup generally increases with the number of cores used
- As a result of overhead and communication, speedup is not linear
  - Theoretically max speedup should be 128x for 128 cores for example, but it is ~70x when searching for one thousand nearest neighbors

Parallel Computing Frameworks

- OpenMP
- MPI (Message Passing Interface)
- CAS (Compare Exchange) used for Lock-Free data structure

Background: KNN

Basics:
- Common algorithm used for classification
- Classifies label of new points based on labels of *k* nearest neighbors

Steps for simple algorithm:
- Iterate over all points and calculate the distance from the current point to the target point
- Create a list of the k nearest neighbors based on the distance
  - May create priority queue based on distance
  - May create an array of k points making sure to sort as points are inserted
- Find the most common label from the *k* nearest neighbors and assign that label to the target point

Using KD Tree for KNN Search:
- Start from root
- Traverse the tree to the section where the new point belongs
  - Based on feature currently being analyzed
- Add node to queue
- Check to see if there are “better” nodes on the other side

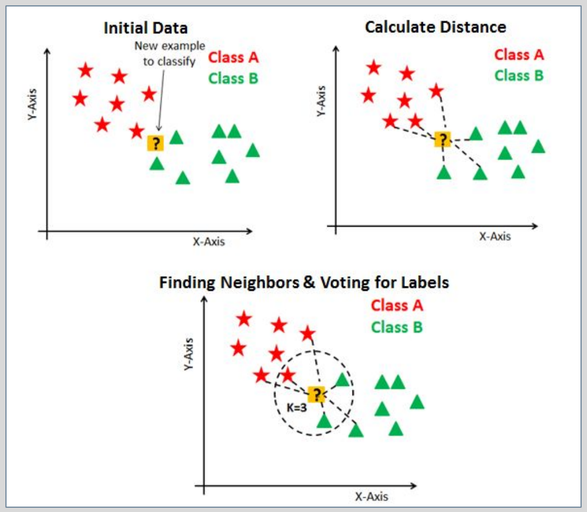


Figure 4: KNN Algorithm
https://rekhavsrh.medium.com/all-about-knn-algorithm-6b35a18c2b15

References

https://rekhavsrh.medium.com/all-about-knn-algorithm-6b35a18c2b15

https://core.ac.uk/download/pdf/19529722.pdf

https://dl.acm.org/doi/pdf/10.1145/361002.361007

https://www.cs.cmu.edu/~ckingsf/bioinfo-lectures/kdtrees.pdf

https://www.researchgate.net/publication/2948052_KNN_Model-Based_Approach_in_Classification

Doreen Valmyr (dvalmyr)
Neelansh Kaabra (nkaabra)