Parallel Algorithms for Constructing Range and Nearest-Neighbor Searching Data Structures

Pankaj K. Agarwal    Kyle Fox    Kamesh Munagala
   Abhinandan Nath
Motivation
Motivation
Motivation

- Some public datasets on Amazon's AWS:
  - 2000 US Census (200 GB)
  - Common Crawl Corpus (web crawl data for 5 billion webpages, 541 TB)

- Advances in LiDar and other remote sensing technologies producing huge spatial data sets
  - Libre Map Project (includes detailed topographic maps of all 50 states)
  - OpenStreetMap
  - ...
What we want to do?

- Build indices to query huge data efficiently.
Tackling Huge Data

• Earlier - I/O model of computation [Aggarwal and Vitter, 1988].

• Now
  – Data too big to fit on one machine, distributed across multiple machines.
  – Process in parallel.
  – Many frameworks available.
    • MapReduce [Dean and Ghemawat, 2008]
    • Hadoop [White, 2012]
    • Pregel [Malewicz et al., 2010]
    • ...
Model of Computation

- RAM model and I/O model not sufficient.

- **Massively Parallel Communication (MPC) model**
  [Beame et al., 2013]
  - Captures salient features of MapReduce and others while hiding low-level details.
MPC Model

- **m**: number of machines.
- **n**: input distributed across m machines.
- **s = n/m**: each machine has $O(s)$ storage.

Assume $s = n^\alpha$, for $0 < \alpha < 1$. 
MPC Model

- Computation proceeds in rounds. In each round
  - Each machine computes on local data.

- Communication b/w machines occurs b/w rounds.

- \#messages sent/received by any machine in a round bounded by O(s).
Performance Measures

- # rounds of computation: \( R \).

- Running time: \( T \).
  - \( t_{\beta r} \) = running time of machine \( \beta \) in round \( r \).
  - \( T = \Sigma_r \max_\beta t_{\beta r} \).

- Total work: \( W = \Sigma_r \Sigma_\beta t_{\beta r} \).
Previous Work

- Extensive work in databases on algorithms for query processing in MapReduce and its variants
  
  [Lee et al., 2012; Qin et al., 2014; Malewicz et al., 2010; Beame et al., 2013 ...].

- MapReduce implementations for analyzing and querying spatial and geometric data
  
  [Eldawy et al., 2013, 2015; Arabi et al., 2014; ...] - however, no provable performance guarantee.
Our Work

- Build distributed variants of the following classical data structures for
  - Range searching
    - Kd-tree [Bentley, 1975]
    - Range tree [Bentley, 1980]
  - Nearest-neighbor searching
    - BBD-tree [Arya et al., 1998]
Our Results

- Kd-tree:
  - Construction:
    - $O(1)$ rounds
    - $O(s \log s)$ time
    - $O(n \log n)$ work
  - Query:
    - $O(1)$ rounds
    - $O(s^{1-1/d} + k')$ time
    - $O(n^{1-1/d} + k)$ work
Our Results

- Range tree:
  - Construction:
    - $O(1)$ rounds
    - $O(s \log^d s)$ time
    - $O(n \log^d n)$ work
  - Query:
    - $O(1)$ rounds
    - $O(\log^d n + k)$ time and work
Our Results

- BBD-tree:
  - Construction:
    - $O(1)$ rounds
    - $O(s \log s)$ time
    - $O(n \log n)$ work
  - Query:
    - $O(1)$ rounds
    - $O(\log n)$ time and work
This talk

- Focus on distributed kd-tree.
- Briefly discuss BBD-tree.
Key Ingredient: Random Sampling

- Use random sample to build top few levels of tree.

- Use partial tree to partition the input points and recurse.
MPC primitive : Partition

П : Spatial subdivision
I : Set of machines
P : Set of points stored on I

Partition and redistribute P so that each machine stores the points of only one cell.

• Can be done in O(1) rounds, O(s log s) time and O(|P| log |P|) work.
MPC primitive: Partition

Before Partition
MPC primitive : Partition

After Partition

[Diagram of a partition with red and blue dots, followed by images of three computer screens with different colored dots arranged in a grid pattern]
Range Searching
Range Searching
Kd-tree

Balanced binary tree, represents a hierarchical decomposition of space.
Kd-tree

Balanced binary tree, represents a hierarchical decomposition of space.
Kd-tree

Balanced binary tree, represents a hierarchical decomposition of space.
Kd-tree

Balanced binary tree, represents a hierarchical decomposition of space.
Anwering range queries using kd-tree

• Start at the root.

• Let □ be query rectangle and □ be box associated with node of kd-tree.
  – □ and □ do not intersect : do nothing.
Answering range queries using kd-tree

- Start at the root.
- Let □ be query rectangle and □ be box associated with node of kd-tree.
  - Report all points inside □.
• Start at the root.

• Let □ be query rectangle and □ be box associated with node of kd-tree.
  - Recurse on children of □.
Query time

- \( k \) : # points inside query rectangle

Query time is \( O(n^{1-1/d} + k) \).
Storage

Use $r$ so that

$$O(r^2 \log n) = O(s^{1/2}).$$

- The top $\log_2 r$ levels are stored on one machine.
- Each node at that level points to the machine that stores the root of the recursive kd-tree on points of the node.
Building kd-tree in MPC

1. Compute a $O(s^{1/2})$-sized random sample of input points and store it on one machine.
1. Compute a $O(s^{1/2})$-sized random sample of input points and store it on one machine.
1. Compute a $O(s^{1/2})$-sized random sample of input points and store it on one machine.

2. Build partial kd-tree on the sample with $r$ leaves (or $\log_2 r$ levels) and store it on the same machine.
Building kd-tree in MPC

1. Compute a $O(s^{1/2})$-sized random sample of input points and store it on one machine.

2. Build partial kd-tree on the sample with $r$ leaves (or $\log_2 r$ levels) and store it on the same machine.

3. Partition all the points using the kd-tree and Partition primitive and recurse in parallel on each piece of partition.
Building kd-tree in MPC

1. Compute a $O(s^{1/2})$-sized random sample of input points and store it on one machine.

2. Build partial kd-tree on the sample with $r$ leaves (or $\log_2 r$ levels) and store it on the same machine.

3. Partition all the points using the kd-tree and Partition primitive and recurse in parallel on each piece of partition.

Each piece has $O(n/r)$ points [Li et al., 2001] with high probability!!
Analysis

- Input size reduces by factor of $O(r)$, so # levels of recursion $O(\log r \cdot n) = O(1)$.

- Each level takes $O(1)$ rounds, $O(s \log s)$ time and $O(n \log n)$ work.

- In total $O(1)$ rounds, $O(s \log s)$ time and $O(n \log n)$ work.

- Final height of the tree is $\log_2 n + O(1)$. 
Query procedure

- Run query on top subtree
Query procedure

- Run query on top subtree
  - takes time and work \(O(s^{1-1/d}).\)
Query procedure

- Run query on top subtree
  - takes time and work \( O(s^{1-1/d}) \).

- Recurse for each leaf of top subtree whose box intersects query rectangle.
Query performance

k : # points inside query rectangle.
k' : max # points reported by a machine.

- # rounds : $O(1)$
- Time : $O(s^{1-1/d} + k')$
- Work : $O(n^{1-1/d} + k)$ – work-optimal if each point is allowed to be stored exactly once!!
BBD-tree

- Induces hierarchical partitioning using annular regions.
- Similar sampling techniques can be used to construct distributed BBD-trees.
Future work

- Using random sampling for geometric graph problems.

- Parallelizing geometric and topological data analysis methods like persistent homology.

- Extending techniques to metric spaces without any embedding provided.
Thanks!!