Parallel DBSCAN Clustering on LiDAR Point Cloud with OpenMP and CUDA

Ethan Ky (etky), Nicholas Beach (nbeach)

March 27, 2024

1 Summary

We are writing a parallel implementation of DBSCAN clustering on LiDAR point clouds, using OpenMP and CUDA. Analysis will involve comparing the implementations and studying scalability across different data distributions. A parallel renderer will be written to visualize the clustering results as they are being computed.

2 URL

https://github.com/eky198/dbscan-point-cloud

3 Background

Density-based spatial clustering, or DBSCAN, is a clustering algorithm which serves as an alternative to K-means clustering. One advantage is that it is much less sensitive to outliers, and performs much better on data distributions with nested patterns. As such, it is often used to pre-process LiDAR point clouds, which vary significantly in data density, for 3D object detectors and other computer vision tasks. An efficient, parallel implementation of the algorithm would be very useful for real-time vision tasks as are often used in autonomous vehicles.

A sequential algorithm is given below:

- While there are unlabelled points:
  - Get a random unlabeled point p.
  - Get neighbor set $N(p)$ of points with distance of at most ϵ from p.
  - If $|N(p)| \geq$ minPts, p is labelled as a core point.
  - If $N(p)$ contains a core point, p is labelled as a border point.
  - Otherwise, point p is labelled as a noise point.
  - If p is a core point, for each neighbor $q \in N(p)$:
    - If q has no label, repeat the previous steps.
    - If q is a core point or border point, assign it to the same cluster as p.

4 The Challenge

We anticipate thread interaction and communication will pose a challenge for us. Given initial points are selected randomly and the number of threads can scale to be greater than the number of clusters, merging threads' clusters will have to take place which could create divergent execution. Alternative implementations may also have non-trivial synchronization challenges, especially if we are using more efficient neighbor search strategies by constructing k-d trees or other data structures.

Another challenge we anticipate is updating the render in real time with the DBSCAN algorithm. The standard approach to rendering graphics is to do all of the computation, update a graphic, and then render it. However, one of our ideas was to update the rendered image as the points are being clustered to show the progression of the DBSCAN. A point cloud has tons of data points, and updating each point individually from potentially different threads presents a huge challenge. There will need to be communication between threads on what data they are trying to update and synchronization to batch each thread's update together. We feel this could be an interesting demonstration and visualization of how the algorithm operates in parallel across different parallel implementations, but we understand its scope does not go beyond that so it will not be a main focus for the project.

5 Resources

We plan on using the GHC machines to compare our parallel implementations, as these contain NVIDIA GeForce RTX 2080 GPU's. We also plan on starting our code base from scratch. The point cloud data will likely be pulled from the KITTI vision benchmark suite. Once we shift our focus to the renderer, we will use a C++ graphics library to help us display the point cloud as it gets clustered.

6 Goals and Deliverables

- What We Plan to Achieve
  - Implement a working sequential version of DBSCAN.
  - Implement a working OpenMP parallel version of DBSCAN. We would like to achieve speedup compared to the sequential version.
  - Implement a working CUDA parallel version of DBSCAN. We would like to achieve better speedup than the OpenMP implementation. This is reasonable as GPU hardware should be better at massive parallel computation.
  - Compare speedup of our implementations across a varied selection of point cloud inputs.
  - Implement a parallel renderer to visualize the point cloud as it is being clustered. This could be used to showcase our system at the poster session.
- What We Hope to Achieve
  - For the poster session, we would like to prepare a side-by-side visualization which compares the clustering times of our implementations.
  - If time permits, we could attempt to perform clustering of point cloud streams. This would modify the clusters in real-time as points are added or removed frame-by-frame.

7 Platform Choice

We will implement this project using C++ with both CUDA and OpenMP. CUDA lends itself well to parallel processing of large-scale point cloud data due to GPU execution. The OpenMP API gives good baseline performance on a CPU, which we can compare GPU performance to.

8 Schedule

- Week 1 (March 31 - April 6): We will read academic papers to familiarize ourselves with the algorithm and possible optimizations.
- Week 2 (April 7 - April 13): We will write a baseline sequential and parallel OpenMP implementation, and collect results on the GHC machines. In doing so, we will formulate efficient data structures and computation schemes that would be built upon in the CUDA implementation.
- Week 3 (April 14 - April 20): We will write a CUDA implementation and collect results on the GHC machines, possibly exploring several parallization schemes.
- Week 4 (April 21 - April 27): We will split time between optimizing the parallel implementations, and writing a parallel renderer to visualize the point cloud clustering results.
- Week 5 (April 28 - May 4): We will write the report and prepare for the poster session.