Parallel Multilevel Clustering of Large Data Sets for Interactive Visualization Applications

Akshay Jain
(N10757112)

Introduction

In this project, I implement a custom clustering algorithm to produce an output view suitable for remote visualizations.

Interactive visualization of points in space of the web is quite challenging when the data is large. The present rendering technologies on the browser support only ~10k points on a present day laptop. Network speed is also a barrier to sending over view updates in realtime as the client might not be able to perform the operations on points in time. Generating static images on the server and sending it over is also not an option as the client needs to be aware of the different objects in the view to get a non-laggy user experience.

The solution I have developed consists of showing two types of objects to the user: Points and Clusters. When a user zooms into a cluster, more points and smaller clusters which make up the larger cluster are shown. This strategy limits the total number of points and clusters ever rendered on the client screen to below what the browser can easily operate on. As long as the user goes easy on his zooming, enough time
would be available between zoom changes to recalculate clusters on data modifications or to calculate lower levels of the visualization.

Algorithm

Input: multiple files containing 2D points. (each line has tab separated x and y coordinates)

1. N MPI processes are started each of which load an equal number of files in memory and calculate the local min and max for x and y.
2. An allreduce is used to calculate global min and max and let all processes know about these values.
3. Each process maps their local data to [0-100, 0-100] and fills a count array that determines number of points closest to each grid point.
4. AllGather is used to communicate this count array to all processes.
5. Using the gathered N count arrays, each process calculates the number of points for separate set of grid points.
6. If this count exceeds the threshold for 1 or more grid points, it marks them. Marked grid-points are sent to the root.
7. Root collates and broadcasts the marked grid points.
8. Each process sends over the data-points mapped to one of the marked grid points to their respective allocated processes using sends and receives (from the earlier count array each process already knows how much data it will receive from each of the other processes).
9. The points mapped to a marked grid-point are written out to files, one for each marked grid point.

10. The unclustered points and the marked grid points with their sizes are then written out to output view files.

11. The view files are used by the server to render the visualization.

The same program is run on the cluster data files to generate the next view and smaller clusters.

Output quality

The output generated on local machine was fed into a web visualization system to generate the following outputs:
Figure 1: Number of points: 1,000 ; Threshold: 50
The output generated on running the program for another iteration on the cluster details output of the first file.

Figure 2: Two levels of clustering

Output from 50k points on the local machine:

Figure 3: Number of points: 50k; Threshold:100
Analysis

Local run-times:
Ubuntu VM on Windows 8
2 GB RAM
4 CPUs

<table>
<thead>
<tr>
<th>Total number of points</th>
<th>Number of input files</th>
<th>Number of lines per file</th>
<th>MPI processes</th>
<th>Clustering Threshold</th>
<th>Time Taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>100,000</td>
<td>10</td>
<td>10,000</td>
<td>10</td>
<td>10</td>
<td>1.196 sec</td>
</tr>
<tr>
<td>1,000,000</td>
<td>10</td>
<td>100,000</td>
<td>10</td>
<td>10</td>
<td>1.593 sec</td>
</tr>
<tr>
<td>10,000,000</td>
<td>10</td>
<td>1,000,000</td>
<td>10</td>
<td>10</td>
<td>12.968 sec</td>
</tr>
<tr>
<td>100,000,000</td>
<td>10</td>
<td>10,000,000</td>
<td>10</td>
<td>10</td>
<td>1m 29.324 sec</td>
</tr>
</tbody>
</table>

Runs on Stampede:

<table>
<thead>
<tr>
<th>Total number of points</th>
<th>Number of input files</th>
<th>Number of lines per file</th>
<th>MPI processes</th>
<th>Clustering Threshold</th>
<th>Time Taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>2,560,000,000</td>
<td>10</td>
<td>256,000,000</td>
<td>256</td>
<td>50</td>
<td>11 seconds</td>
</tr>
<tr>
<td>25,600,000,000</td>
<td>10</td>
<td>2,560,000,000</td>
<td>256</td>
<td>50</td>
<td>27 seconds</td>
</tr>
<tr>
<td>256,000,000</td>
<td>10</td>
<td>25,600,000</td>
<td>64</td>
<td>50</td>
<td>2.9 seconds</td>
</tr>
<tr>
<td>25,600,000</td>
<td>10</td>
<td>2,560,000</td>
<td>64</td>
<td>50</td>
<td>1.9 seconds</td>
</tr>
<tr>
<td>2,560,000</td>
<td>10</td>
<td>256,000</td>
<td>64</td>
<td>50</td>
<td>1.5 seconds</td>
</tr>
<tr>
<td>256,000</td>
<td>10</td>
<td>25,600</td>
<td>64</td>
<td>50</td>
<td>1.4 seconds</td>
</tr>
</tbody>
</table>

Conclusion:

We get pretty nice speeds with 64 MPI processes on stampede on about 256m points, viable to be used as the backend for a web visualization application. For larger data-sets, although the clustering occurs pretty fast, it is not viable for a near real-time web visualization app.
Future Work:

Here are two ideas that could help speed-up this implementation further so as to provide the same speeds with lesser number of MPI processes. Also this could bring the billion point range in reach.

- Optimize network usage further: Though the implementation is very parallel right now, decreasing the network communication by encoding and decoding count arrays into single strings or floating points could help.
- Local chucking before data-transfer: a higher density threshold can be defined and used locally for chunking before doing the distributed clustering on these chunks. This will reduce the clustering quality a little, but could result into a high performance improvement.