Connected Component Labeling using MPI

Kun Lin
CSE633
Table of Content

- Problem Description
- Sequential Algorithm
- Parallel Approach
- Results and Comparisons
- Conclusion
Problem Description

Connected-component labeling, also known for region extraction or region labeling is a graph theory problem.

- Goal is to detect unique region in a binary image where each unique region is given an unique label.
- Each foreground pixel can be considered an vertex, vertices are neighbors if they’re one pixel spacing away. We could have four-connected neighbors or eight connected neighbors.
Real world applications

Labeling CT cross-section

Application on clustering scene
Two Pass algorithm (sequential)

Processor will scan through the image two times (row major)

First pass:
- If the pixel $x$ is a foreground pixel, check its neighbors that is above $x$ and on the left of $x$.
- If two neighbors are background pixels, then $x$ will have a new unique label, or if only one of them is a foreground pixel with an existing label, then $x$ will be assigned the same label.
- If two neighbors are both foreground pixels and have different labels, we will take the minimum but set up an equivalent list.

Second pass:
- Re-labeling each foreground pixel based on its lowest equivalent list.
Parallel Approach

- Divide the image by rows, where each processor get some row intervals of image
- Each Processor locally compute two pass algorithm on the local image
- All processor will pass only the neighboring row result to root processor, then root processor will compute a global equivalent list and broadcast the list to all processor
- Then all processors will perform second pass that will re-label all its local necessary pixels
Equivalent list example

- Currently using dictionary as data structure to store equivalent list
  - Where 1=> 2; 1=> 3 will store as Dictionary [1] => [2, 3]

Some possible cases

```
1  2  3
2  3  4
1
2
3
```

```
1  2, 3
1  2, 3, 4
```

```
Individual result from each node

Combined result

Labeling 16 x 16 size image, divide data into 4 node, each have size 4x16 data

Boundary labels send to root node to compute global equivalent list then bcast the result for relabeling.
Testing parallel runs

- Using MPI Library and UB CCR academic cluster
- Programs are ran in range of 1 - 128 CPUs, tested on image size $2^7 \times 2^7$ and $2^8 \times 2^8$
- Data could be divide in many ways, for this experiment they are divided by row, which is straightforward to keep track of index and corresponding communication between processors
## Results of 128 X 128 Graph

<table>
<thead>
<tr>
<th># Processors</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.51</td>
</tr>
<tr>
<td>4</td>
<td>0.384</td>
</tr>
<tr>
<td>8</td>
<td>0.153</td>
</tr>
<tr>
<td>16</td>
<td>0.25</td>
</tr>
<tr>
<td>32</td>
<td>2.286</td>
</tr>
<tr>
<td>64</td>
<td>10.13</td>
</tr>
<tr>
<td>128</td>
<td>39.20</td>
</tr>
</tbody>
</table>
## Results of 256 X 256 Graph

<table>
<thead>
<tr>
<th># Processors</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>87</td>
</tr>
<tr>
<td>4</td>
<td>14.102</td>
</tr>
<tr>
<td>8</td>
<td>2.246</td>
</tr>
<tr>
<td>16</td>
<td>1.3</td>
</tr>
<tr>
<td>32</td>
<td>3.37</td>
</tr>
<tr>
<td>64</td>
<td>13.113</td>
</tr>
<tr>
<td>128</td>
<td>61.065</td>
</tr>
</tbody>
</table>
Speed up / Scaling

Let two node’s runtime be our base case

<table>
<thead>
<tr>
<th># Processors</th>
<th>Scaling</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>6.169</td>
</tr>
<tr>
<td>8</td>
<td>38.736</td>
</tr>
<tr>
<td>16</td>
<td>66.923</td>
</tr>
<tr>
<td>32</td>
<td>25.82</td>
</tr>
<tr>
<td>64</td>
<td>1.425</td>
</tr>
</tbody>
</table>

Speed Up vs. # of Node
Conclusion

- Parallel Component labeling showed faster run time than sequential
- Using number of node in range from 4 - 32 we can see a significant speed up as input image size increases, as we adding more nodes, we tends to see longer run time that might be cause by too much processor communication
- Graph tested for this project are randomly generated
  - Real world image’s pixel tends to be more cluster and separate, which will work well with this parallel implementation
Reference:

- R. Miller and L. Boxer, Algorithms Sequential and Parallel: A Unified approach
- Connected-component labeling - wikipedia
  https://en.wikipedia.org/wiki/Connected-component_labeling
- Parallel Programming with MPI For python
- Image 1 Slide 3 http://k-sience.blogspot.com/2017/06/object-counting-using-connected.html
- Connected Component - Udacity - youtube-channel
Questions?