Connected Component Labeling using MPI

Kun Lin

CSE633

University at Buffalo The State University of New York

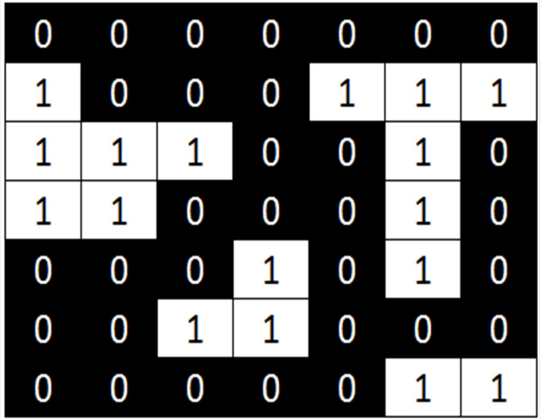
Table of Content

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

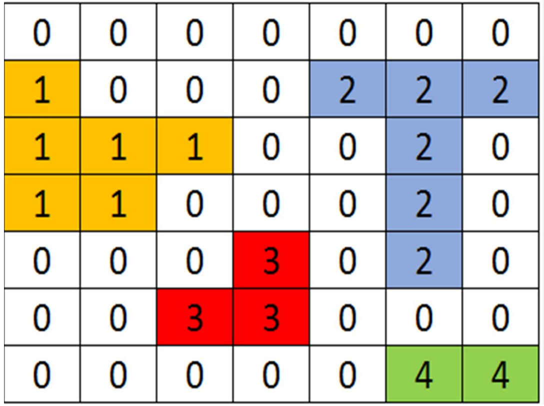
Problem Description

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

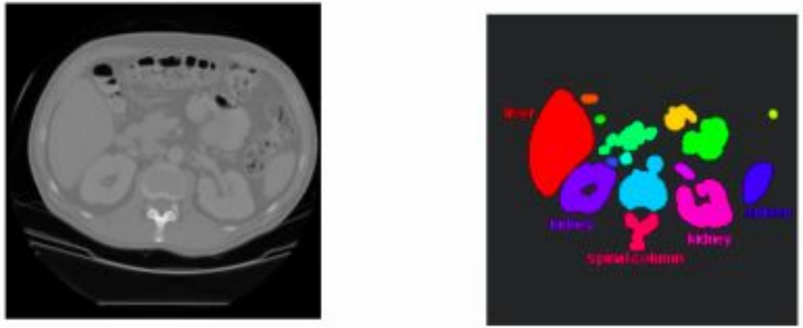


0	0	0	0	0	0	0
1	0	0	0	1	1	1
1	1	1	0	0	1	0
1	1	0	0	0	1	0
0	0	0	1	0	1	0
0	0	1	1	0	0	0
0	0	0	0	0	1	1

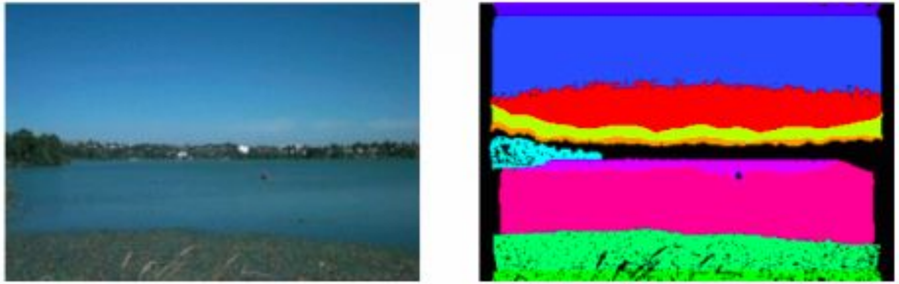


0	0	0	0	0	0	0
1	0	0	0	2	2	2
1	1	1	0	0	2	0
1	1	0	0	0	2	0
0	0	0	3	0	2	0
0	0	3	3	0	0	0
0	0	0	0	0	4	4

Real world applications



Labeling CT
cross-section



Application on
clustering scene

Two Pass algorithm (sequential)

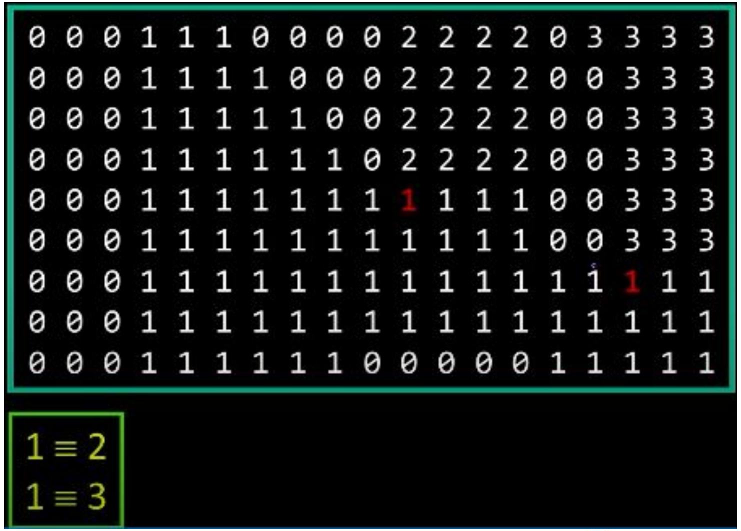
Processor will scan through the image two times (row major)

First pass:

- If the pixel x is a foreground pixel, check the its neighbors that is above x and on the left of x.
- If two neighbor are background pixel, then x will have a new unique label, or if only one of them is foreground pixel with existing label, then x will be assign same label
- If two neighbor are both foreground pixel and have different label - we will take the min but set up a equivalent list

Second pass:

- Re-labeling each foreground pixel based on its lowest equivalent list



```
0 0 0 1 1 1 0 0 0 0 2 2 2 2 0 3 3 3 3
0 0 0 1 1 1 1 0 0 0 2 2 2 2 0 0 3 3 3
0 0 0 1 1 1 1 1 0 0 2 2 2 2 0 0 3 3 3
0 0 0 1 1 1 1 1 1 0 2 2 2 2 0 0 3 3 3
0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 3 3 3
0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 3 3 3
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 1 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1
```

$1 \equiv 2$

$1 \equiv 3$

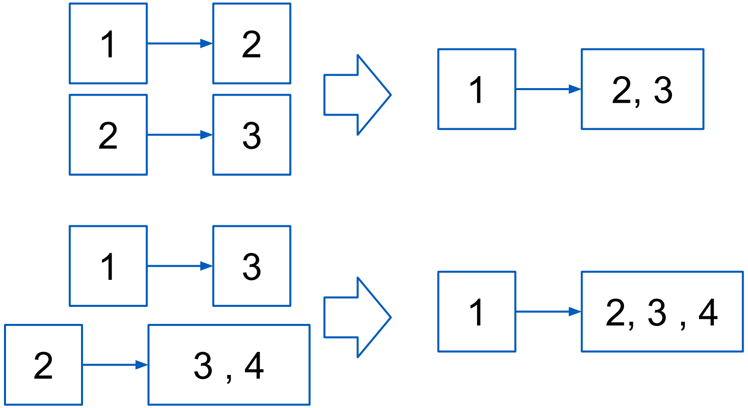
Parallel Approach

- 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

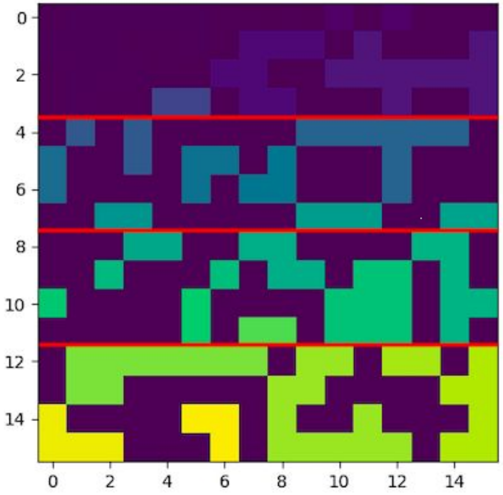
Equivalent list example

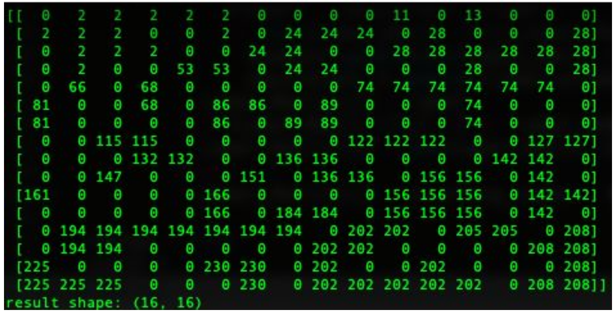
- Currently using dictionary as data structure to store equivalent list
    - Where 1=> 2; 1=> 3 will store as Dictionary [1] => [2, 3]

Some possible cases



Individual result from each node



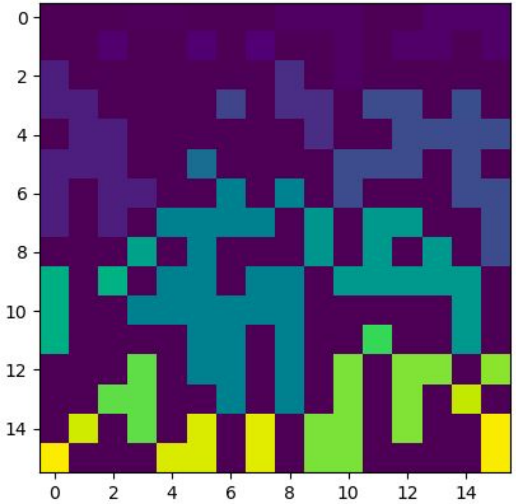


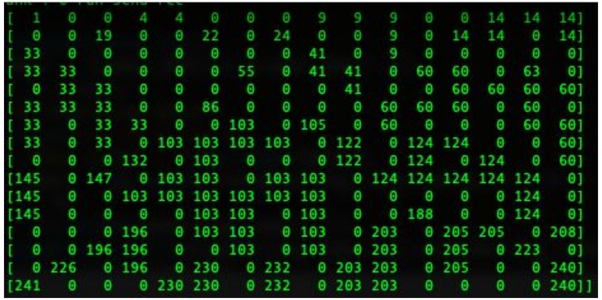
```
[[  0   2   2   2   2   2   0   0   0   0  11   0  13   0   0   0]
 [  2   2   2   0   0   2   0  24  24  24   0  28   0   0   0  28]
 [  0   2   2   2   0   0  24  24   0   0  28  28  28  28  28  28]
 [  0   2   0   0  53  53   0  24  24   0   0   0  28   0   0  28]
 [  0  66   0  68   0   0   0   0   0  74  74  74  74  74  74   0]
 [ 81   0   0  68   0  86  86   0  89   0   0   0  74   0   0   0]
 [ 81   0   0   0   0  86   0  89  89   0   0   0  74   0   0   0]
 [  0   0 115 115   0   0   0   0   0 122 122 122   0   0 127 127]
 [  0   0   0 132 132   0   0 136 136   0   0   0   0 142 142   0]
 [  0   0 147   0   0   0 151   0 136 136   0 156 156   0 142   0]
 [161   0   0   0   0 166   0   0   0   0 156 156 156   0 142 142]
 [  0   0   0   0   0 166   0 184 184   0 156 156 156   0 142   0]
 [  0 194 194 194 194 194 194 194   0 202 202   0 205 205   0 208]
 [  0 194 194   0   0   0   0   0 202 202   0   0   0   0 208 208]
 [225   0   0   0   0 230 230   0 202   0   0 202   0   0   0 208]
 [225 225 225   0   0   0 230   0 202 202 202 202 202   0 208 208]]
result shape: (16, 16)
```

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

Combined result





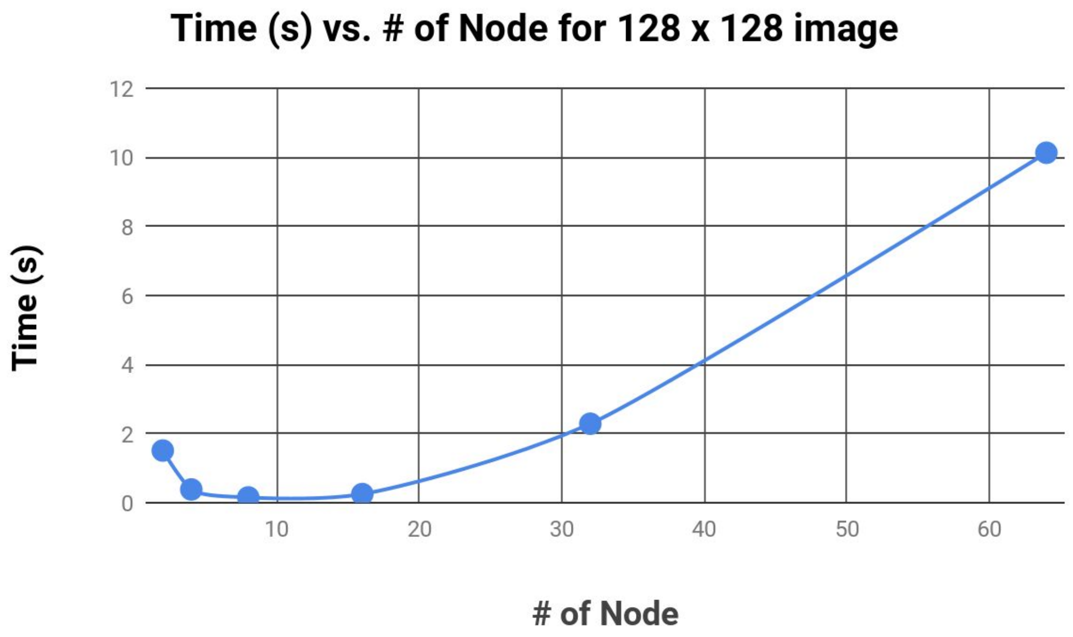
```
[   1   0   0   4   4   0   0   0   9   9   9   0   0  14  14  14]
[   0   0  19   0   0  22   0  24   0   0   9   0  14  14   0  14]
[  33   0   0   0   0   0   0   0  41   0   9   0   0   0   0   0]
[  33  33   0   0   0   0  55   0  41  41   0  60  60   0  63   0]
[   0  33  33   0   0   0   0   0   0  41   0   0  60  60  60  60]
[  33  33  33   0   0  86   0   0   0   0  60  60  60   0  60   0]
[  33   0  33  33   0   0 103   0 105   0  60   0   0   0  60  60]
[  33   0  33   0 103 103 103 103   0 122   0 124 124   0   0  60]
[   0   0   0 132   0 103   0   0   0 122   0 124   0 124   0  60]
[145   0 147   0 103 103   0 103 103   0 124 124 124 124 124   0]
[145   0   0 103 103 103 103 103 103   0   0   0   0   0 124   0]
[145   0   0   0   0 103 103   0 103   0   0 188   0   0 124   0]
[   0   0   0 196   0 103 103   0 103   0 203   0 205 205   0 208]
[   0   0 196 196   0   0 103   0 103   0 203   0 205   0 223   0]
[   0 226   0 196   0 230   0 232   0 203 203   0 205   0   0 240]
[241   0   0   0 230 230   0 232   0 203 203   0   0   0   0 240]]
```

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 x 2^7 and 2^8 x 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

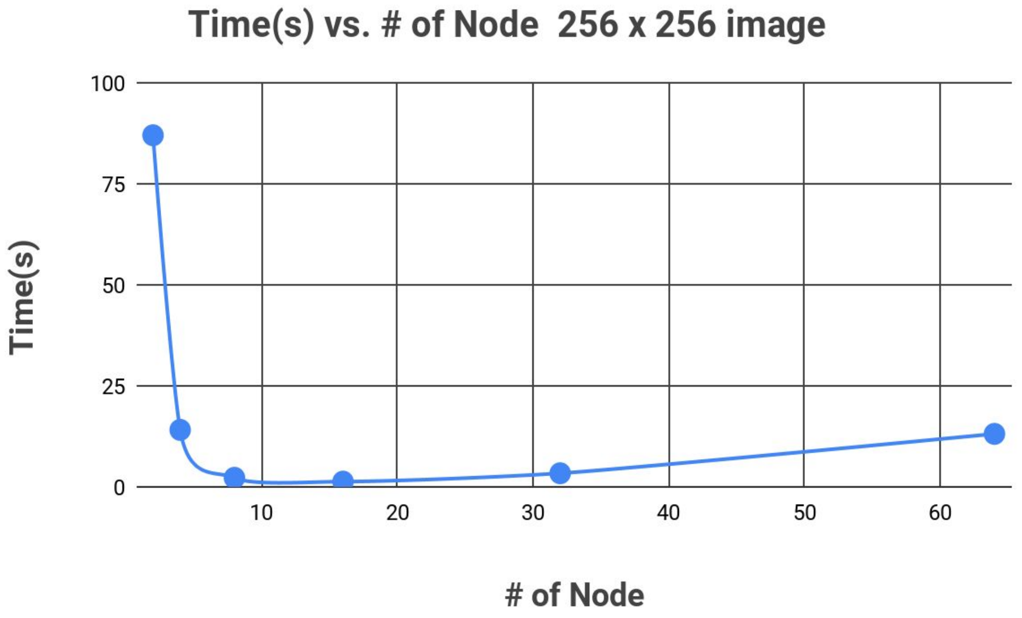
Results of 128 X 128 Graph

# Processors	Time (s)
2	1.51
4	0.384
8	0.153
16	0.25
32	2.286
64	10.13
128	39.20



Results of 256 X 256 Graph

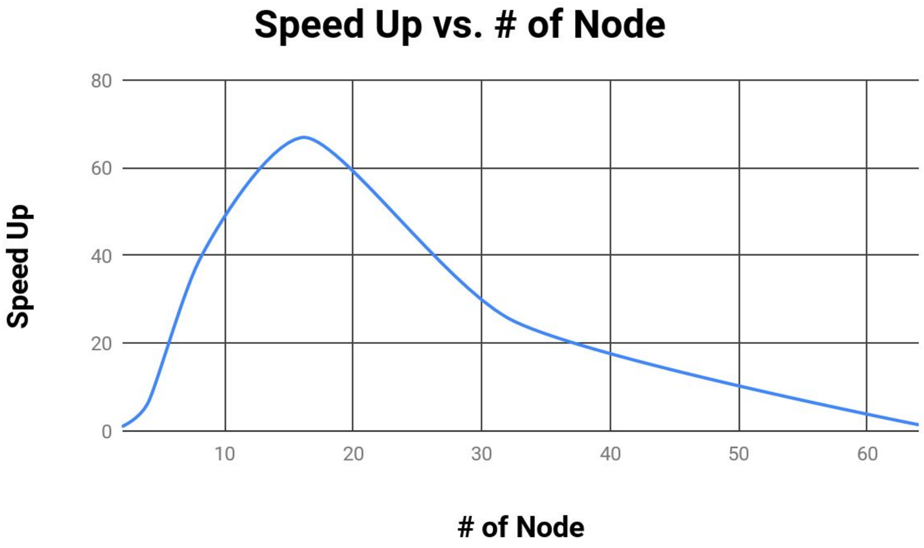
# Processors	Time (s)
2	87
4	14.102
8	2.246
16	1.3
32	3.37
64	13.113
128	61.065



Speed up / Scaling

Let two node's runtime be our base case

# Processors	Scaling
2	1
4	6.169
8	38.736
16	66.923
32	25.82
64	1.425



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-componenet labeling - wikipedia https://en.wikipedia.org/wiki/Connected-component_labeling
- Parallel Programming with MPI For python https://rabernat.github.io/research_computing/parallel-programming-with-mpi-for-python.html
- Image 1 Slide 3 http://k-sience.blogspot.com/2017/06/object-counting-using-connected.html
- Connected Component - Udacity - youtube-channel
- http://www.cs.utexas.edu/~grauman/courses/378/slides/lecture3_full.pdf

Questions?