Image Compression using k-Means Clustering

CSE633 Spring ‘19

Instructor- Dr. Russ Miller

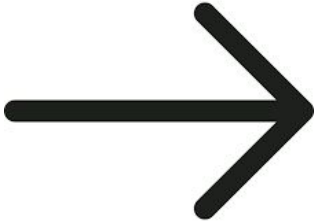
By Utkarsh Bansal

Outline

- Problem Definition
- k-Means for Clustering
- Implementation
- Results
- Observations
- Challenges
- References

Problem Definition

Compressing an Image using k-Means Clustering.



28929 unique colors

20 unique colors

k-Means for Clustering

1. Randomly select k points to be cluster centers.
2. For each point in the data set, put it in the cluster which has its center closest to the point.
3. Calculate new cluster centers by taking means of all points in a cluster.

Repeat 2 and 3 until convergence or exit condition reached.

Implementation

Creating Dataset from Image (Serial)

- Read the image using OpenCV for Python.
- Append the R, G, and B values of the pixels to a string one by one.
- Saving the string to a .txt file.

Parallel k-Means

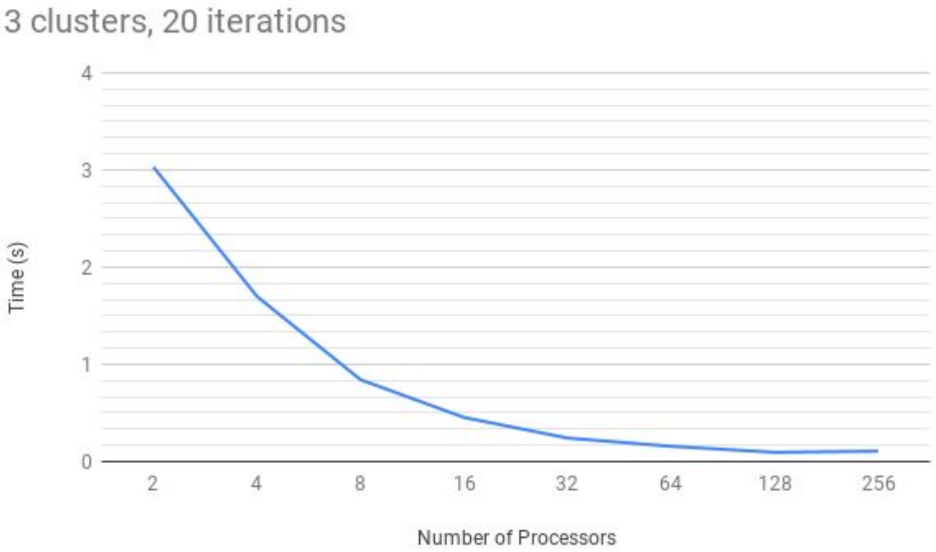
- Consider N data points and P processors.
- Assign N/P data points to each processor using .txt files.
- Node 0 randomly chooses k points as cluster means and broadcasts them.
- Each processor for each of its points, finds the cluster to which the point belongs.
- Recalculate local sums for each cluster in each processor.
- Send all local sums for each processor to processor 0 to find the global means.
- Repeat the clustering for number of iterations.
- Save the cluster means of the final iteration.

Compressed Image Formation (Serial)

- Read the file with final cluster means.
- Read the image.
- For each pixel, determine the cluster it belongs to.
- Overwrite the pixel value to the cluster mean.
- Save the resulting image.

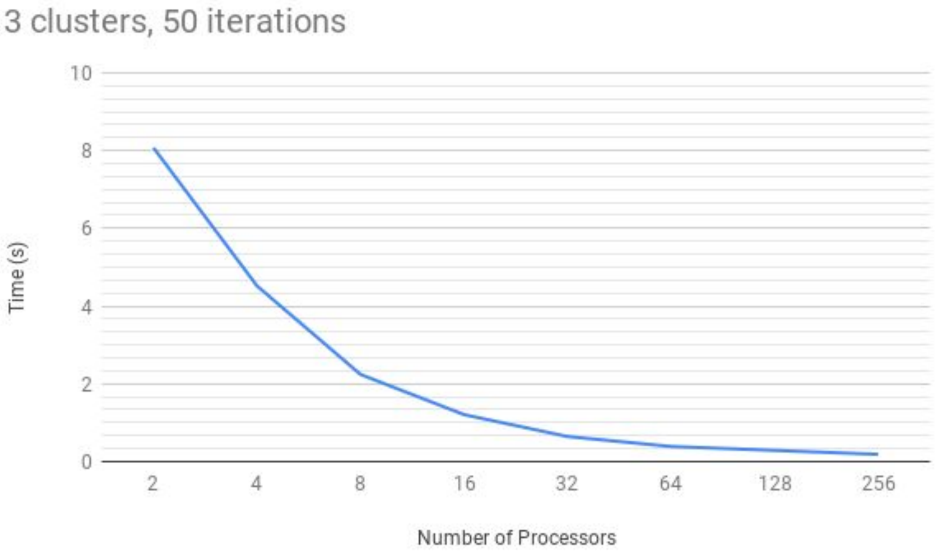
Results

Time Analysis for 3 Clusters



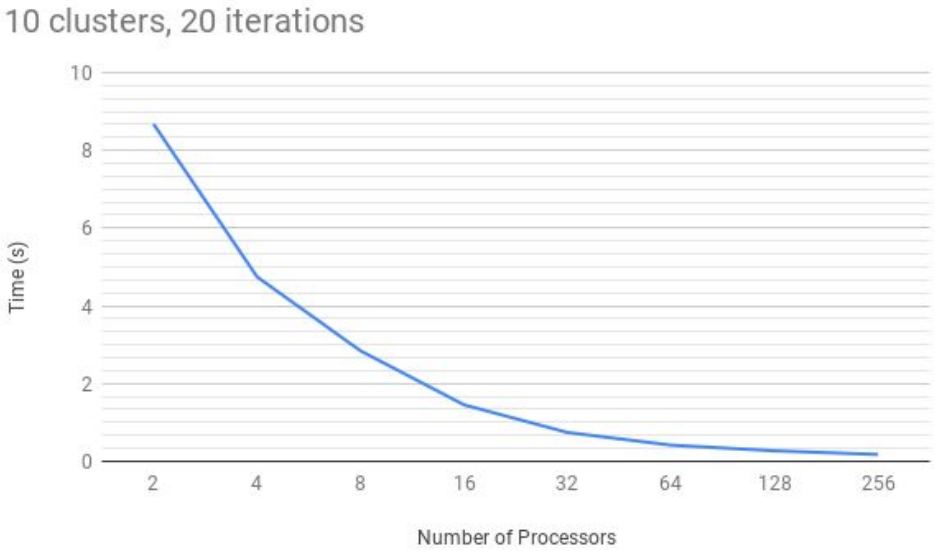
Number of Processors	Time in seconds
2	3.04
4	1.69
8	0.88
16	0.46
32	0.25
64	0.16
128	0.15
256	0.15

Time Analysis for 3 Clusters



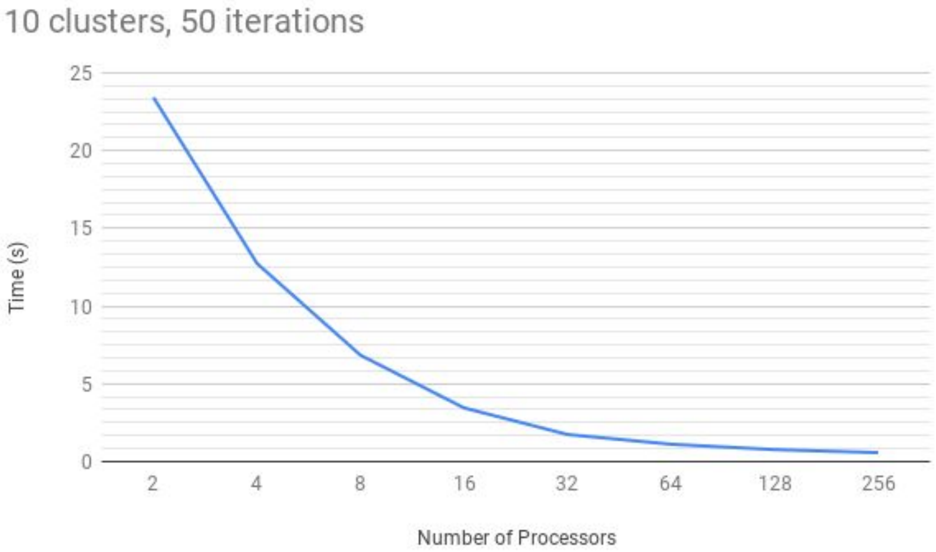
Number of Processors	Time in seconds
2	8.17
4	4.52
8	2.24
16	1.20
32	0.64
64	0.38
128	0.28
256	0.20

Time Analysis for 10 Clusters



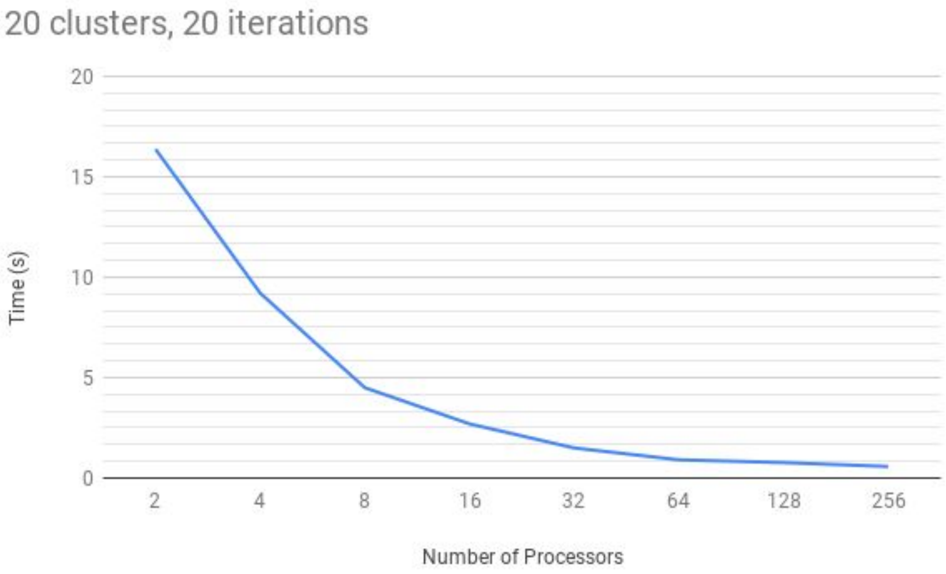
Number of Processors	Time in seconds
2	8.32
4	4.33
8	2.49
16	1.36
32	0.71
64	0.60
128	0.33
256	0.23

Time Analysis for 10 Clusters



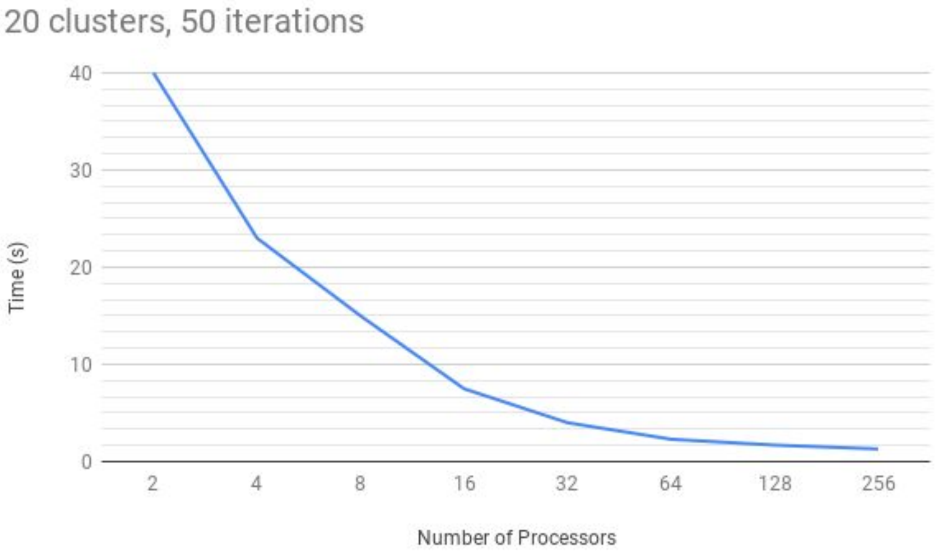
Number of Processors	Time in seconds
2	23.43
4	12.55
8	6.86
16	3.42
32	1.71
64	0.89
128	0.81
256	0.76

Time Analysis for 20 Clusters



Number of Processors	Time in seconds
2	16.62
4	9.23
8	4.62
16	2.52
32	1.36
64	0.85
128	0.79
256	0.73

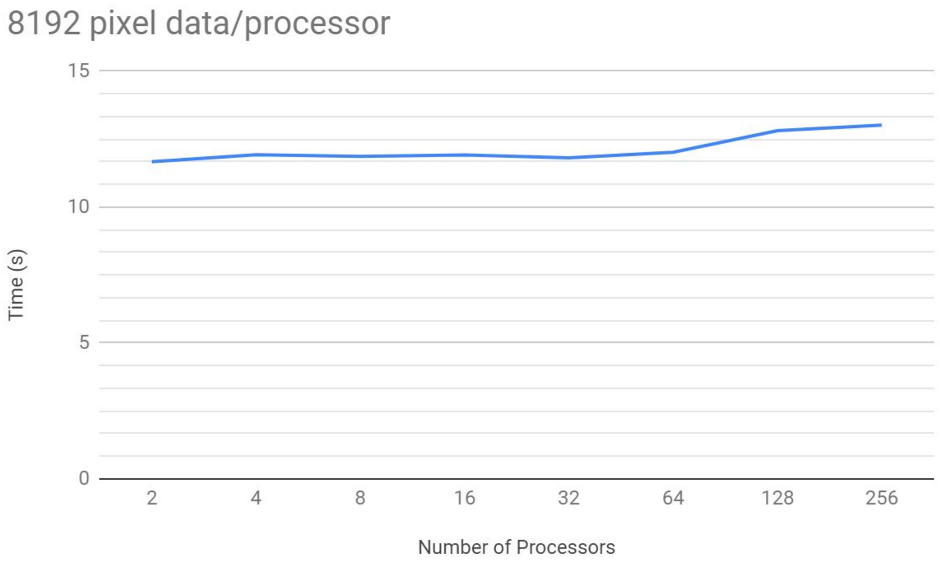
Time Analysis for 20 Clusters



Number of Processors	Time in seconds
2	39.98
4	22.33
8	11.82
16	8.00
32	4.17
64	2.33
128	1.70
256	1.60

We had varying data/processor till now. Now keeping the data/processor constant at data 256*32 = 8192 pixel data.

Time Analysis for Constant Data/Processor



Number of Processors	Time in seconds
2	11.67
4	11.93
8	11.87
16	11.92
32	11.81
64	12.02
128	12.81
256	13.02

Output Images

3 colors

10 colors

20 colors

Observations

- Significant decrease in time is observed only upto 32 processors.
- For the input size used, using more than 32 processors is not practical.
- Cost of communication when the number of processors is large, significantly overshadowed the benefit of increasing the number of processors further.
- When we have constant data per processor, cost of communication is reflected by the increase in time with increasing number of processors.

Challenges

- The input .txt files and the output images had to be created serially.
- Ran into insufficient memory errors when the input files were too big. (>4000*4000 pixels)
- Running the script on 256 nodes took a long time (around 11 hours) and I ran into a lot of issues getting the output right.

References

- Algorithms Sequential & Parallel: A Unified Approach (Dr. Russ Miller, Dr.Laurence Boxer)
- https://ubccr.freshdesk.com/support/solutions/articles/13000026245-tutorials-and-training-documents
- https://www.buffalo.edu/ccr/support/ccr-help.html
- https://mpi4py.readthedocs.io/en/stable/

Thank You!