Image colour quantization using K-means clustering algorithm

Sourabh Suri, Shashidhar Gopagani, Krishna Khairnar
Department of Electrical Engineering
IIT Bombay, Powai - 400076
sourabh.suri@iitb.ac.in, 193079010@iitb.ac.in, krishnak@ee.iitb.ac.in

Nidhal Hafeez
Department of Energy Science and Engineering
IIT Bombay, Powai - 400076
nidhal.h@iitb.ac.in

***Abstract*—Image colour quantization is the method that reduces the number of distinct colours utilized in a picture, typically with the intention that the new image ought to be as visually similar as doable to the first image. Colour quantization is essential for showing pictures with several colours on devices that can solely display a restricted range of colours, typically because of memory limitations, and also permits efficient compression of certain forms of pictures for storage and transmission. In this project, we implement colour quantization using the most popular algorithm of K-means clustering. We present a performance analysis of serial and parallel execution by OpenMP, CUDA, and MPI platforms. We observe that CUDA's performance exceeds other platforms' performance in terms of time taken (sec).**

***Index Terms*—Image colour quantization, K-means clustering algorithm, Serial programming, Parallel Programming, OpenMP, CUDA, MPI, profiling**

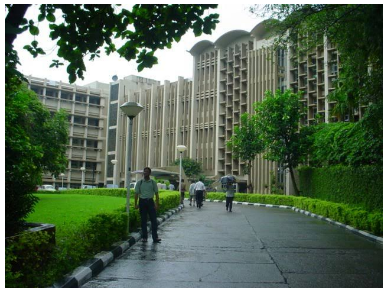
I. Introduction

Colour quantization decreases the number of distinct colours in an image while maintaining the image's visual appearance, as illustrated in Fig. 1. It is imperative to implement colour quantization efficiently using parallel programming techniques because of its applicability in numerous graphics and image processing techniques, primarily for mobile devices and legacy hardware.

Pixel is the smallest element of an image. At any point, the value of a pixel corresponds to the intensity of the light at that particular location. Pixels in coloured images have associated 24-bits containing (2^{24}) 1,67,77,216 different colours represented as three-dimensional vectors, each vector element with an 8-bit dynamic range, called RGB triplets. A colour palette is a representative smaller set of colours in an image. During colour quantization implementation, each pixel's stored value maps into one of the closest colours in the colour palette. This way, only the values related to the colour palette can be stored, thus requiring lesser space than the original image [1].

We have used k-means clustering to group similar colour values into K clusters, and each pixel value is replaced by the value of the centroid of that cluster. K-Means Clustering is a very effective, easy-to-implementing, and fast machine learning technique for image colour quantization [2].

To obtain higher performance, we investigate various parallel programming techniques and platforms such as CUDA, OpenMP and MPI. Theoretically, all the pixels can be mapped simultaneously; however, for OpenMP and MPI implementation, the number of processors limits the acceleration. In Farivar R. et al., it is shown that with p cores, the first stage of k-means clustering, which involves mapping of the pixels to the cluster centres, can be accelerated from $\mathcal{O}(nk)$ to $\mathcal{O}(nk/p)$ where n is the number of data points and k is the number of clusters [3].


(a) Original

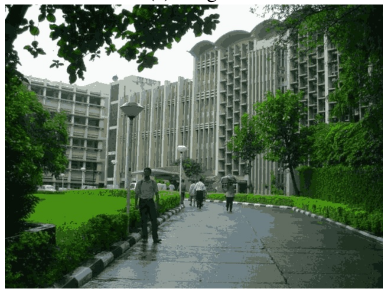

(b) Colour quantized

Fig. 1: Image colour quantiztion illustration

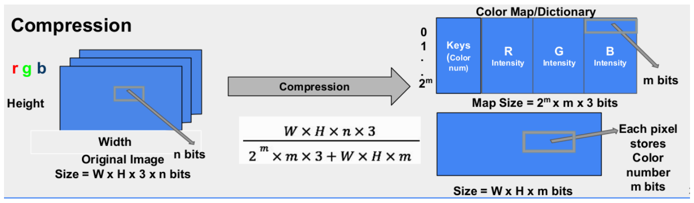

Compression
r g b
Height
Width
Original Image
n bits
Size = W x H x 3 x n bits
Compression
$\frac{W \times H \times n \times 3}{2^m \times m \times 3 + W \times H \times m}$
Color Map/Dictionary
0
1
2^m
Keys (Color num)
R Intensity
G Intensity
B Intensity
m bits
Map Size = 2^m x m x 3 bits
Each pixel stores Color number m bits
Size = W x H x m bits

Fig. 2: Compression Ratio

II. Implementation

In this section, we first describe the sequential and then the parallel implementation of the k-mean clustering algorithm for image colour quantization.

A. *Sequential K-means clustering algorithm*

In image colour quantization, after randomly initializing the K centroids, the iterative algorithm of K-means have primarily two steps [4]:

- Cluster Assignment : In this, we allot the centroid (or cluster centre) index to the pixel values such that the euclidean distance between the pixel and the centroid is minimum. The number of clusters k equals the number of colours desired in the colour quantized image.
- Re-initialising Centroids : In this, we calculate the new cluster centres, by taking the mean of all the pixel values that is allotted to a given cluster.

The k-means algorithm attempts to optimize the objective or cost function by reiterating the above steps until there is zero change in cluster centres.

A similar algorithm, as shown in Fig. 3, is used for other parallelization platforms. First, we convert the 3D input image using OpenCV to a 2D vector where RGB values are stored in a sequence for every pixel. We dynamically allocate memories for input image 2D vector, cluster map and centroid colour pixels dictionary. A cluster map stores the value of all clustered pixels against the centroids. At the same time, the centroid colour pixels dictionary stores the values of RGB intensities against the centroids.

Second, we initialize random positions from the image to K, which represent the number of colours. After initialization, we iterate through all the pixels in the image and try to find the centroid pixel to which it belongs. The similarity between two pixels is measured by using the euclidean distance between the RGB intensities. If we can't find any centroid corresponding to the selected pixel, we will push that pixel as one of the new centroid centres. After all the pixels in the image are mapped to the centroids, we will take the average of all the pixel values (i.e. RGB separately) and assign new centroid pixels according to the result. This process is repeated until there is no change in centroid values. After the convergence, we will form a new quantized image based on K selected pixels.

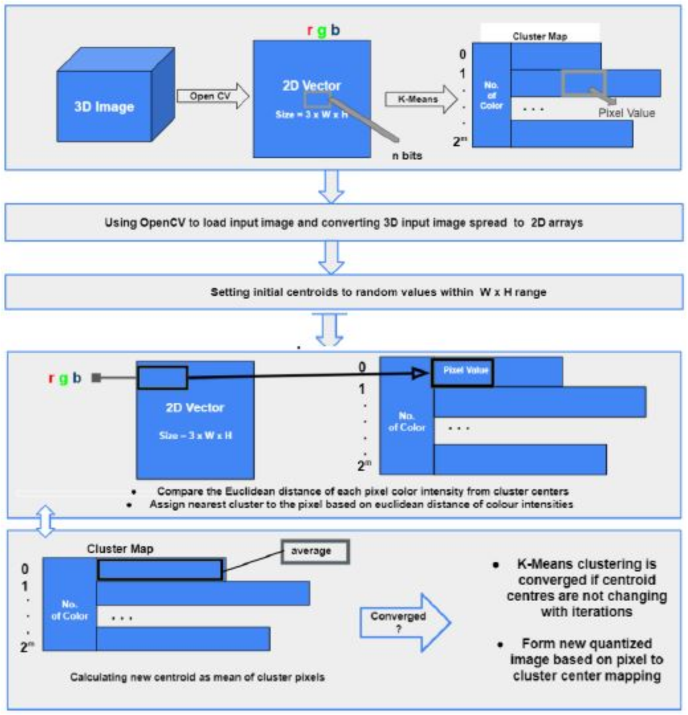

r g b
3D Image
Open CV
2D Vector
Size = 3 x W x h
n bits
K-Means
Cluster Map
0
1
2^m
No. of Color
Pixel Value
Using OpenCV to load input image and converting 3D input image spread to 2D arrays
Setting initial centroids to random values within W x H range
r g b
2D Vector
Size – 3 x W x H
Pixel Value
No. of Color
Compare the Euclidean distance of each pixel color intensity from cluster centers
Assign nearest cluster to the pixel based on euclidean distance of colour intensities
Cluster Map
average
No. of Color
Calculating new centroid as mean of cluster pixels
Converged ?
K-Means clustering is converged if centroid centres are not changing with iterations
Form new quantized image based on pixel to cluster center mapping

Fig. 3: Flowchart depicting the serial implementation of k-mean clustering algorithm for image colour quantization

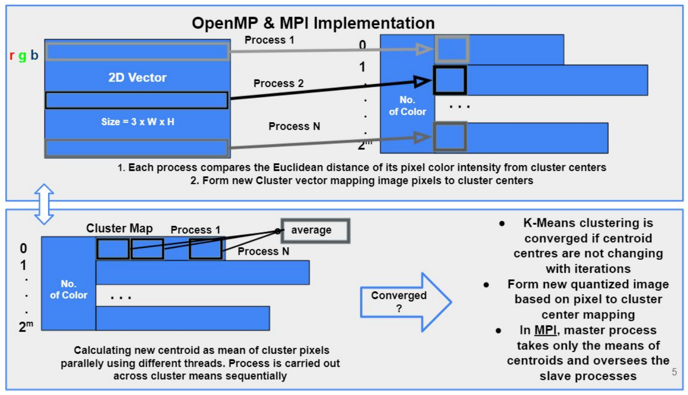

OpenMP & MPI Implementation
r g b
2D Vector
Size = 3 x W x H
Process 1
Process 2
Process N
0
1
2^m
No. of Color
1. Each process compares the Euclidean distance of its pixel color intensity from cluster centers
2. Form new Cluster vector mapping image pixels to cluster centers
Cluster Map
Process 1
average
Process N
No. of Color
Calculating new centroid as mean of cluster pixels parallely using different threads. Process is carried out across cluster means sequentially
Converged ?
K-Means clustering is converged if centroid centres are not changing with iterations
Form new quantized image based on pixel to cluster center mapping
In MPI, master process takes only the means of centroids and oversees the slave processes

Fig. 4: Flowchart depicting the parallelized implementation of k-mean clustering algorithm for image colour quantization using OpenMP and MPI platform

B. *Parallelization*

1) OpenMP & MPI: Here, mapping of selected pixel data points to cluster centroids is done parallelly because this mapping is independent amongst pixels. Thus, there is no sequential dependency. However, for the next step, where the mean of all pixels is done, dependencies are there as the variable average in RAM is common, or we can say they are shared amongst each centroid. Thus, for OpenMP implementation, we calculate the mean for each cluster using reduction, i.e. in parallel. This is done in a sequential way for each cluster. In case of MPI, the master process takes the means of centroids sequentially and oversees the slave processes as shown in Fig. 4.

2) CUDA : For CUDA and MPI implementation, instead of vectors, we use pointers for simplicity during communication or GPU-CPU memory copy instructions. Thus, we load the input image using OpenCV and then convert the 3D input image spread to three(RGB) pointers to 1D arrays. Then we allocate memory for input image RGB pixels, cluster array and centroid colour pixels. Then, We initialize random positions from the image to K, centroids representing the number of colours. Finally, we set grid size and block size based on the width and height of the input image such that the pixel gets a thread for its computation.

After setting the number of threads, we launch K-Means

TABLE I: Time comparison of various platforms on different images with five K values

K	Platform	Image 1	Image 2	Image 3	Image 4	Image 5	Average
4	*Serial*	9.36	24.02	90.88	47.13	36.01	41.48
4	*OpenMP*	3.36	8.16	30.96	15.20	11.89	13.914
4	*MPI*	0.79	0.74	3.17	1.2	3.56	1.892
4	*CUDA*	0.88	0.91	1.14	0.93	0.93	0.958
4	*CR*	17.3084	17.3107	17.3118	17.3107	17.3110	17.31052
8	*Serial*	28.08	68.28	211.36	71.16	96.96	96.168
8	*OpenMP*	6.98	15.42	47.83	30.33	34.88	27.088
8	*MPI*	1.18	7.37	10.59	1.56	27.77	9.694
8	*CUDA*	0.91	0.98	1.27	0.98	1.08	1.044
8	*CR*	11.5381	11.5401	11.5411	11.5401	11.5404	11.53996
16	*Serial*	52.62	228.80	1179.7	135.42	523.75	424.058
16	*OpenMP*	11.63	45.54	248.38	27.85	109.39	88.558
16	*MPI*	2.12	13.52	32.09	7.95	21.83	15.502
16	*CUDA*	0.97	1.27	1.57	1.19	1.37	1.274
16	*CR*	8.6522	8.6546	8.6556	8.6545	8.6548	8.65432
32	*Serial*	345.50	867.85	2129.23	795.77	388.49	905.368
32	*OpenMP*	72.75	167.55	401.41	166.09	74.57	176.474
32	*MPI*	8.88	30.94	89.85	16.37	57.02	40.612
32	*CUDA*	1.37	1.55	2.13	1.52	1.54	1.622
32	*CR*	6.9199	6.9228	6.9242	6.9228	6.9232	6.92258
64	*Serial*	720	1791.51	5462.15	1561.4	1623.71	2231.754
64	*OpenMP*	77.97	332.61	512.59	410.74	432.53	353.288
64	*MPI*	65.68	112.26	193.22	264.71	106.52	148.478
64	*CUDA*	1.15	1.74	2.52	1.74	2.35	1.9
64	*CR*	5.7638	5.7679	5.7698	5.7679	5.7691	5.7677

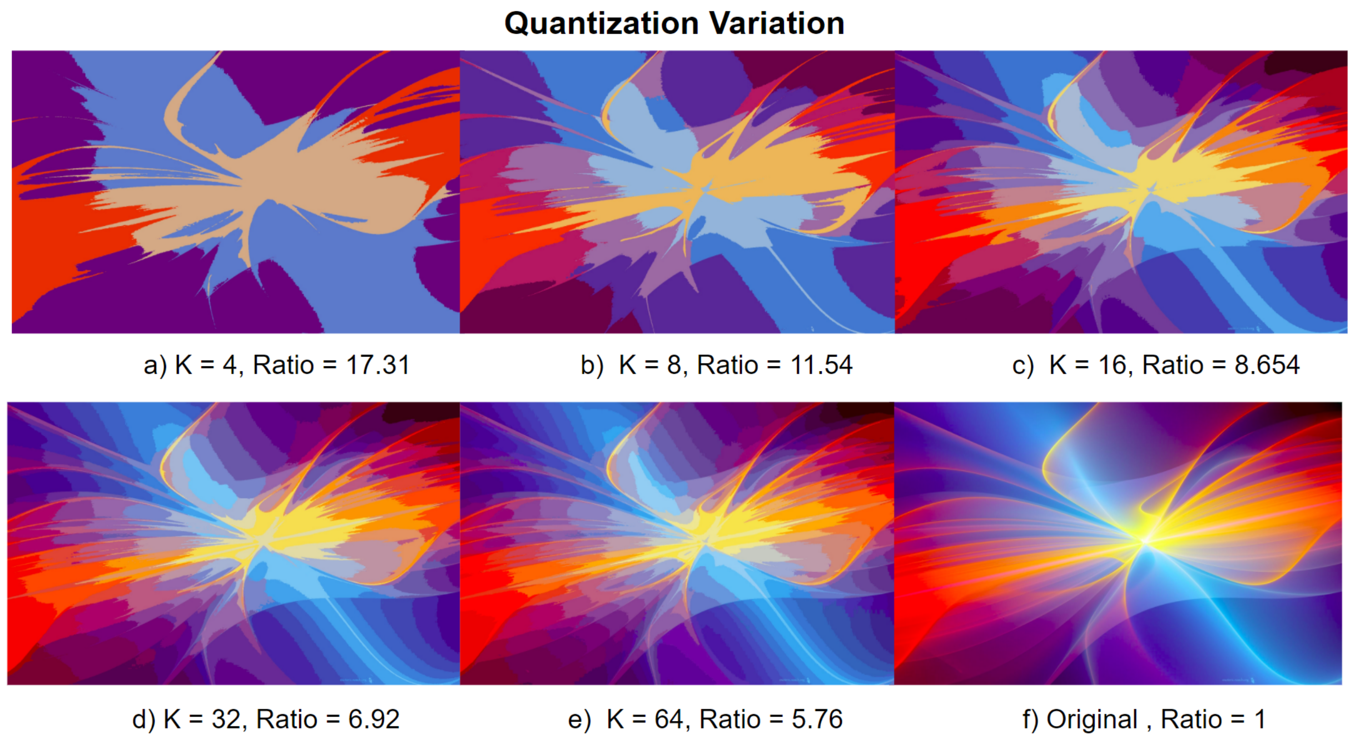
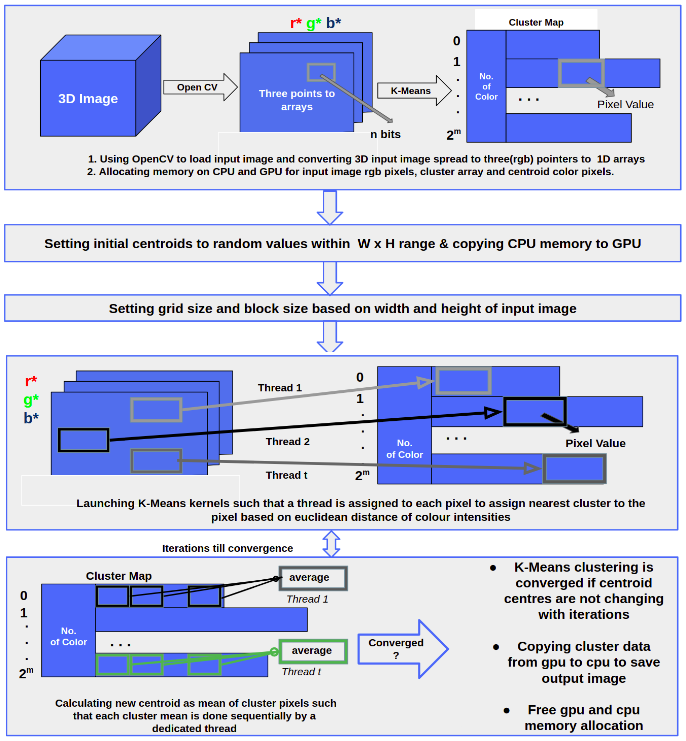


Fig. 8: Variation in colour quantization of an image having different compression ratio

kernels such that a thread is assigned to each pixel which assigns the nearest cluster to the pixel based on a euclidean distance of colour intensities. Since each pixel mapping is independent of each other, the more the number of cores in the GPU helps in reducing the overall time complexity and accelerating the performance.

Since there are dependencies in calculating average as the variable average is a shared variable, we estimate a new centroid as a mean of cluster pixels. A dedicated sum thread adds pixel value to variable average atomically because other sum threads might also be adding to variable average. Then the mean is calculated sequentially by each dedicated thread called mean thread which divides the sum by a number of elements in a cluster. The corresponding flowchart is as shown in Fig. 5.



3D Image
Open CV
r* g* b*
Three points to arrays
n bits
K-Means
Cluster Map
0
1
No. of Color
Pixel Value
2^m
1. Using OpenCV to load input image and converting 3D input image spread to three(rgb) pointers to 1D arrays
2. Allocating memory on CPU and GPU for input image rgb pixels, cluster array and centroid color pixels.
Setting initial centroids to random values within W x H range & copying CPU memory to GPU
Setting grid size and block size based on width and height of input image
r*
g*
b*
Thread 1
Thread 2
Thread t
No. of Color
Pixel Value
2^m
Launching K-Means kernels such that a thread is assigned to each pixel to assign nearest cluster to the pixel based on euclidean distance of colour intensities
Iterations till convergence
Cluster Map
average
Thread 1
average
Thread t
No. of Color
2^m
Converged ?
Calculating new centroid as mean of cluster pixels such that each cluster mean is done sequentially by a dedicated thread
K-Means clustering is converged if centroid centres are not changing with iterations
Copying cluster data from gpu to cpu to save output image
Free gpu and cpu memory allocation

Fig. 5: Flowchart depicting the parallelized implementation of k-mean clustering algorithm for image colour quantization using CUDA platform

III. Results and Conclusion

We have populated the timing and compression ratios of the various platforms in the Table. 1. We are using five different images, and their size is also mentioned. We have compiled for different k values representing the number of different colours for quantization. It can be observed that as number of colour quantization decreases, the compression ratio increases which results in a less storage space requirements.

However, the image's visual quality deteriorates as k is reduced. When we increase the number of clusters, the quality of the image also improves. There is minimal loss of quality by reducing the image with almost all colours to just 64 distinct colours. This concept is used to make images compatible with devices that can display only a few limited colours.

As shown in Fig. 5, an increase in the number of pixels leads to the increased time taken. However, the time is not entirely dependent on size. This is due to the early convergence of the k-means algorithm, leading to a lesser time. The convergence of the algorithm depends on the number of distinct pixels in the original image. The number of iterations depends on the convergence of the algorithm; this cannot be improved by parallel programming - only the computations which occur in each iteration can be made more efficient by executing them in parallel. From Fig. 6, it is evident that CUDA outperforms other platforms mainly because the algorithm is highly parallel. Also, as we go for less number of colour quantizations the compression ratio increases.

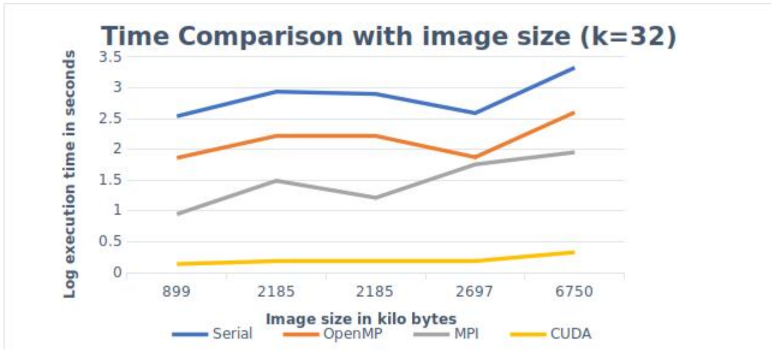


Fig. 6: Time comparison with images' size for K = 32

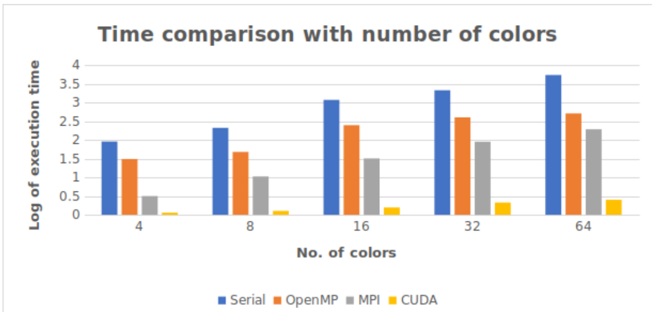


Fig. 7: Time comparison with the number of colours

References

[1] Márquez -de-Silva S., Felipe-Riverón E., Sánchez Fernández L.P. (2008) A Simple and Effective Method of Color Image Quantization. In: Ruiz-Shulcloper J., Kropatsch W.G. (eds) Progress in Pattern Recognition, Image Analysis and Applications. CIARP 2008. Lecture Notes in Computer Science, vol 5197. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-85920-8.

[2] M. E. Celebi, "Effective initialization of k-means for color quantization," in Image Processing (ICIP), 2009 16th IEEE International Conference on. IEEE, 2009, pp. 1649–1652.

[3] R. Farivar, D. Rebolledo, E. Chan, and R. H. Campbell, "A parallel implementation of k-means clustering on gpus." in Pdpta, vol. 13, no. 2, 2008, pp. 212–312.

[4] Keshav Tangri, "Color Quantization for Image Reduction using K-Means Clustering Algorithm", https://medium.com/, Accessed on 12 May 2021.