Image Segmentation using K-Means Clustering

Tzu-Heng Lin
henry19980520@gmail.com
Institute of Computer Science and Engineering
Hsinchu, Taiwan

Fang-Hua Chang
a0912938624@gmail.com
Institute of Data Science and Engineering
Hsinchu, Taiwan

Yu-Ting Tseng
yuting.tseng.cs10@nycu.edu.tw
Institute of Network Engineering
Hsinchu, Taiwan

1 ABSTRACT

Image segmentation is a technique that divides a digital image into various subgroups known as Image segments, which aims to improve downstream processing or analysis of the image by reducing the complexity of the original image. K-means clustering algorithm is one of the most widely used algorithms. However, when dealing with huge datasets with numerous dimensions and clusters, K-means suffers from computational problems. As a result, our research aims to further improve the performance of the k-means algorithm by using computing resources in modern machines and parallel computing.

2 INTRODUCTION

Image segmentation is the most simplest way to compress an image, but nowadays there is a tendency for larger and larger image sizes. There are three of the most common image sizes for the web.
1920×1080 px, 1280×720 px, 1080×1080 px. In addition, we need to consider the parameter k. A larger k means that we need more computations to process this algorithm. Therefore, we would like to use parallel techniques to speed up the overall time, or to give a larger k in the same amount of time.

3 PROPOSED SOLUTION

The K-means clustering algorithm is the most commonly used algorithm because of its ease in implementation. The serial time complexity is $O(h * w * k * i)$, where h is the height of the image, w is the width of the image, k is the number of clusters and i is the number of iterations.

Serial K-means Algorithm:

(1) Random k points are selected as initial centroids.
(2) **Repeat**.
  (a) k clusters are formed by assigning all points to the closest centroids.
  (b) Centroids are recomputed for each cluster.
(3) **Until** the centroids don't change.

For the K-means clustering problem, we try to leverage multithreading techniques via OpenMP, and to achieve this goal, we proposed the following methodology.

Multithreading Algorithm:

p = the number of processor, k = the number of clusters, h = the height of input image, w = the width of input image.

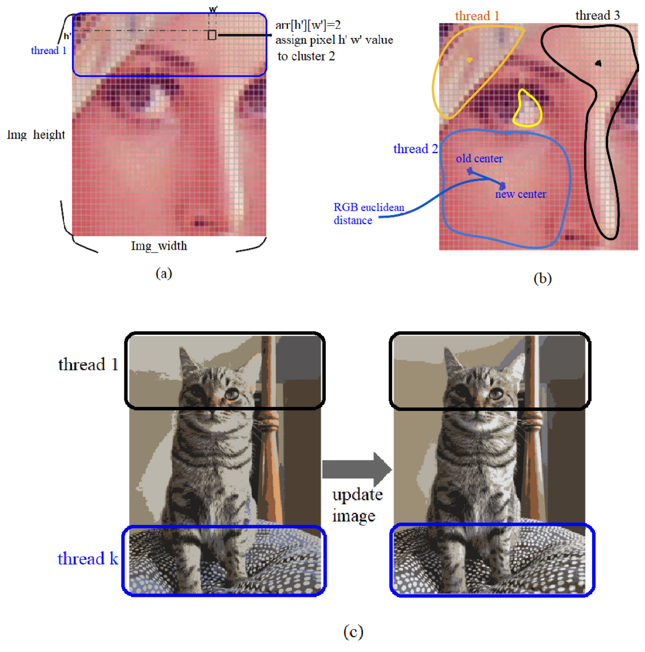

Figure 1: Schematic diagram of our proposed multithreading method.

(1) Random k points are selected as initial centroids.
(2) **Repeat**.
  (a) Assign h/p to each processor to find the associated cluster for each pixel.
  (b) Store the reallocated cluster number k' in a dynamic 2d array of $h * w$ size, which is shown in Fig.1 step(a).
  (c) Assign k/p to each processor to compute the center color euclidean distance, which is shown in Fig.1 step(b).
  (d) Update new center to each cluster.
  (e) meanNewCenter = $(sumupnewCentereuclideandistance)/k$.
  (f) Calculate diffChange = $abs(meanOldCenter - meanNewCenter)$.
(3) **Until** the centroids don't change.
(4) Assign h/p to each processor to write Image, which is shown in Fig.1 step(c).

The time complexity analysis of each step of our proposed method is as follows:
For Figure 1 step (a): We split tasks by image height, the original time complexity $O(h * w * k)$ is reduced to $O(w * k)$ after parallelization.
For Figure 1 step (b): We split tasks by the number of cluster, the original time complexity $O(h * w)$ is reduced to $O(h * w/k)$ after

Figure 2: source image

	size
original	5475 * 3794
large	1920 * 1330
medium	1280 * 887
small	640 * 443

Table 1: Image size

parallelization.
For Figure 1 step (c): We split tasks by image height, the original time complexity $O(h * w)$ is reduced to $O(w)$ after parallelization. Sum up the time complexity of each step and multiply it by iteration time i, the final time complexity of our proposed method is $O(i * (w * k + h * w/k))$, which greatly reduces the time complexity of the serial algorithm.

4 EXPERIMENTAL METHODOLOGY

Environment:

We test our multithreading algorithm by OpenMP in the environment of AMD Ryzen™ Threadripper™ 3960X and Ubuntu 16.04 . It has 24 cores and 48 threads. Based on hardware resources, the number of threads is set to 1 to 48.

Input sets:

Our experiments use the same image, but 4 various size scales, source Image is shown in Fig.2. The sizes of each image are described in table 1. And the cluster number K is all set to 128.

5 EXPERIMENTAL RESULTS

We want to know whether the size of the image will affect the overall execution time of the program, and whether the size of the image will have an impact when doing parallelization.

First, these experiment results are running on the same image but different sizes, since different images will influence the convergence round. Images with more complex colors will require more

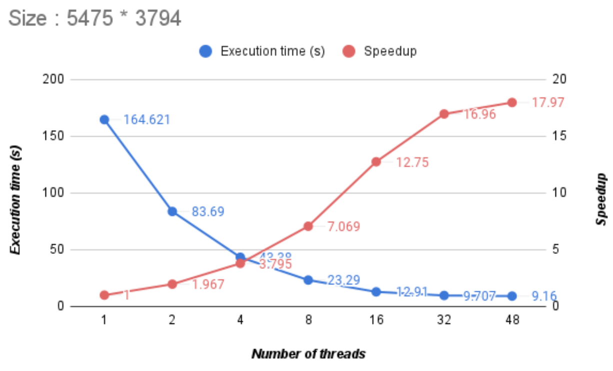

Figure 3: Speedup and Execution curve of original size.

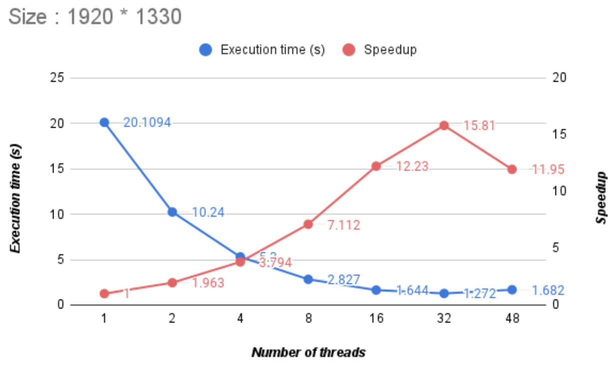

Figure 4: Speedup and Execution curve of large size.

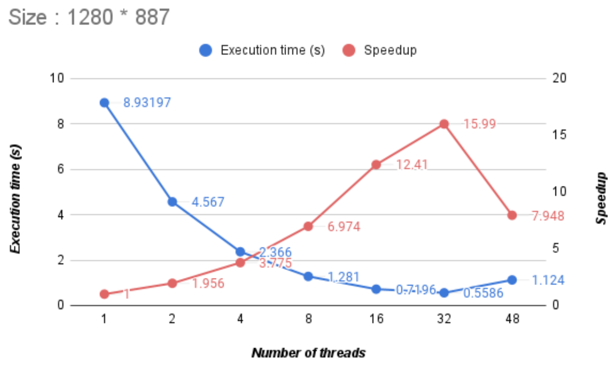

Figure 5: Speedup and Execution curve of medium size.

convergence iterations than images with simpler colors.

According to Figure 3 to Figure 6, the computation time decreases as the image size decreases because we need to perform calculations on each pixel, so the number of pixels will determine the critical cost of the algorithm’s operation.

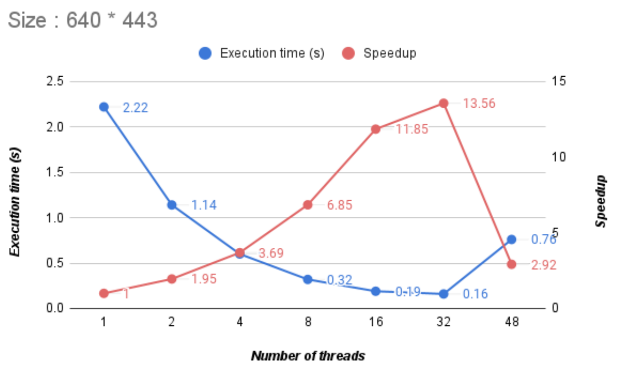

Figure 6: Speedup and Execution curve of small size.

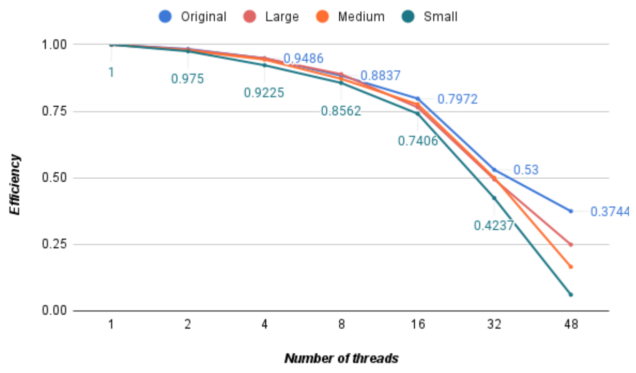

Figure 7: Efficiency curve of each size.

After parallelization, we also want to know how much the overall program is optimized by the parallelization method we used to accelerate and the efficiency of this parallelization method. Therefore, in addition to the speedup, we also calculated the efficiency under different numbers of threads.

From Figure 7, it can be seen that when the number of threads increases to 32, the speedup and efficiency will significantly drop because we need to rebuild the local image for each thread and the rebuilding cost becomes more and more obvious as the number of threads increases.

Because there are three main functions in k-means, we used different parallelization methods to accelerate the overall execution time of the program for these three functions. We want to know the impact of different parallelization methods on this program, so we also recorded the time percentage of these three functions under different numbers of threads.

According to Figure 8 and Figure 9, the time proportion of *"findAssociatedCluster"* which is the step (a) of fig.1 decreases as the number of threads increases, but the time proportion of *"applyFinalClusterToImage"* which is the step (c) of fig.1 increases as the

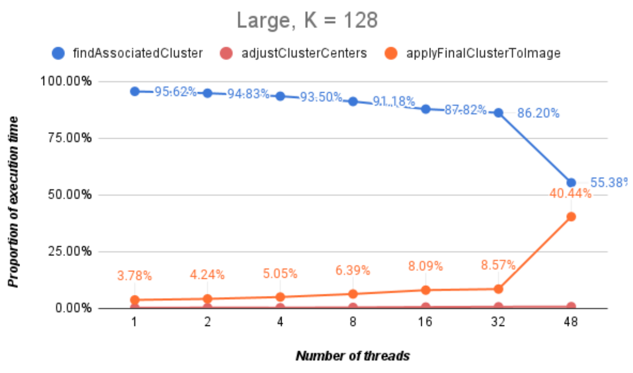

Figure 8: The proportion of each function with a large size.

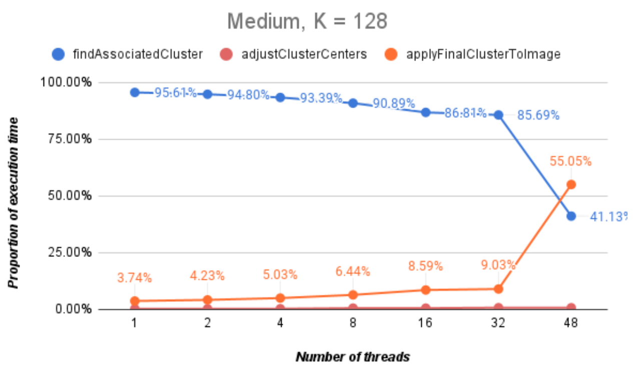

Figure 9: The proportion of each function with a medium size.

number of threads increases.

In the parallelized method of *"applyFinalClusterToImage"* which is the step (c) of fig. 1 , we assign a local_image of the same size as the original image to each thread in order to avoid different threads accessing the same object. However, when the number of threads increases, the cost of rebuilding becomes more and more obvious.

6 RELATED WORK

Existing work [1] divides the image into 2, 4, or 6 blocks, and each part is processed in parallel using k-means clustering to identify clusters within the image. The process involves assigning each instance in the image to the cluster with the nearest centroid, and reassigning the centroid if it is not the closest to the instance. After all the parts have been processed, the image is reconstructed and further calculations are performed.

The difference with us is that the task assigned to their entire method thread is a part of the image. However, based on what the thread does, the thread's task will be divided up into different units. For example, when adjusting to a new cluster, the task per thread

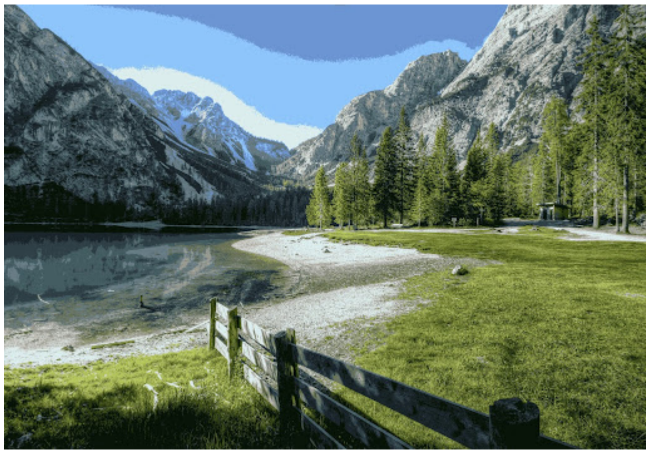

Figure 10: Image segmentation k = 32

is the unit of cluster, which is shown at fig.1 step(b).

The advantage of our method is particularly obvious in reassigning the new cluster part. When the previous work reassigned a new cluster, each thread saw only part of the picture, so the final regenerated picture will have a clear cutting line, even if the calculation result of serial Kmeans is the same cluster. For Bose's work, It is assigned to different threads, so it will be assigned to different clusters. However, our approach fixes this weakness.

7 CONCLUSIONS

(1) As the number of threads increases, the parallelization method that divides by rows obtains better speedup and efficiency than the parallelization method that divides by clusters.
(2) The cost of reconstruction will become more and more apparent as the number of threads increases.

Future direction:

In the *"applyFinalClusterToImage"* function, performance should be better if it is possible to parallelize without assigning a local_image to each thread and also avoiding different threads accessing the same object during rebuilding.

8 REFERENCE

[1] S. Bose, A. Mukherjee and Madhulika "Parallel image segmentation using multi-threading and k-means algorithm." in IEEE International Conference on Computational Intelligence and Computing Research, 2013.
[2] https://ieeexplore.ieee.org/abstract/document/6724171
[3] https://ieeexplore.ieee.org/abstract/document/6240695