# An Implementation of K-Means Image Segmentation on Massively Parallel GPU Architecture

Owen Tanner Wilkerson

Image segmentation is essential for object and boundary detection in today's applications of computer vision. One implementation of image segmentation is through the naturally NP-hard approach of k-means clustering. Thankfully, through heuristic algorithms, we can reduce k-means clustering down to a NP problem that is non-optimal. In general, especially in boundary detection problems, the non-optimal nature of k-means clustering is not a debilitating issue. Yet, the run-time complexity of heuristic k-means clustering is still too large for most of the big data or real-time applications of computer vision to be ran serially general purpose processors. Yet, the k-means algorithm executes the same set of operations on independent elements within a set of data. This property makes the algorithm a prime candidate for investigation of its effectiveness when implemented on SIMD (Single Instruction Multiple Data) machine architectures. The most widely accessible SIMD machine is a standard graphics processing unit (GPU), which is found in most general computing machines.

Additional Key Words and Phrases: Image Segmentation, Algorithms, k-mean

**ACM Reference Format:**

Owen Tanner Wilkerson. 2018. An Implementation of K-Means Image Segmentation on Massively Parallel GPU Architecture. *ACM Trans. Appl. Percept.* 0, 0, Article 0 (October 2018), 11 pages.
DOI: 0000000.0000000

## 1. INTRODUCTION

Image Segmentation is one of the most powerful tools for preprocessing images for applications within computer vision. In basic terms, applying Image Segmentation to an image creates a version of the image that is less complex than the original. This simplified version of the target image makes differentiating between separate objects and surfaces much easier for further image processing algorithms down a given processing pipeline. This proceeding processing pipeline is application specific and could range from detecting software that determines where some mountains are

within a picture to a pipeline that does edge detection on the moving components of a piece of heavy machinery.

For the extent of this report, let us think about Image Segmentation of a picture as looking at the RGB values of each pixel, group the pixels that have numerically close RGB values together, and then reassigning the RGB values of grouped pixels to the average of the RGB values their assigned grouping. This can drastically reduce the detail in an image depending on how many grouping you want to create. Given the right number of groupings, it is possible to eliminate just enough detail that will make objects easier to differentiate from other nearby objects. Yet, forming these grouping fast, efficiently, and effectively is no easy task. K-means clustering is a potential candidate for solving this grouping problem.

## 2. BASIC K-MEANS

K-means is a type of clustering algorithm. This means that k-means will group n data points together into k groups where each value is assigned to a cluster that has a mean value closest to the true value of the observation. A mean value of a cluster is just the mean value of all observations residing within the cluster [Trevino 2016]. The stage at which a data point is assigned to a cluster is commonly referred to as the 'labeling' stage. K-means can be broken down into the following phases:

(1) Seeding cluster origins.

(2) Assign label to data points.

(3) Update cluster orgin.

(4) Repeat items 2 and 3 until no data points are being assigned to a different cluster.

(1) is done by either choosing k random points among the given dataset to act as the initial cluster origins or through some more advanced techniques that attempts to intelligently choose initial cluster origins. For this investigation of k-means, the randomized seeding method was chosen. (2) is the stage where a data point is assigned to a cluster. During this stage, you calculate the Euclidean distance from the data point to the origin of each cluster. The point is then assigned to the cluster it is closer to with respect to Euclidean distance. During stage (3), a clusters origin is redefined to reflect the data points that is not encloses. The new origin is simply the mean of all enclosed data points of a given cluster. Then, noted by (4), you continue steps (2) and (3) until no points are being reassigned to a new cluster; at which point, it is considered that you have converged on a solution [Trevino 2016].

### 2.1 Considering Basic K-means' Complexity

For every data point during the labeling phase (2), the Euclidean distance must be calculated from each data point, to k cluster origins. On sequential CPU cases, the execution time of the labeling phase is on the complexity $O(nk)$. Yet, the cluster origin

update phase in the same environment only requires a time complexity only slightly above linear at $O(n + k)$. If a ratio of these two phases' complexities is considered, $O(nk/(n + k)) \approx O(k)$ for $n >> k$. Now consider was compute ***ONLY*** the labeling phase across p processing elements. Applying this change to the analysis of the phase complexity ratio from before, we acquire $O((nk/p)/(n + k)) \approx O(k/p)$ for $n >> k$. So, by only offloading one phase onto a set of p processing elements, we can reduce our approximate complexity by a factor of p [Sirotković et al. 2012]. Additionally, the calculation of the Euclidean distance from a single data point to each origin of k clusters is completely independent of the execution of the same operation for all other data points. This data point independence is only because recalculation of the cluster origins is held off until all data points have been labeled. Therefore, you are effectively applying the same set of instruction across multiple pieces of data that happen to reside within the same dataset. So, a SIMD parallel architecture is the obvious solution for tackling this parallelization problem.

## 3. THE GPU-BASED PARALLEL K-MEANS ALGORITHM PSEUDOCODE

A few small notations need to be defined before observing the algorithm pseudocode. The code has a population of k clusters, denoted by C. A single cluster is denoted by $c_j$ where $0 \leq j \leq k$ and $c_j \in C$. The image to be segmented has a population of m pixels, denoted by X. A single pixel is denoted by $x_i$ where $0 \leq i \leq m$ and $x_i \in X$. $x_i$ has a label, $l_i$, that matches it to its nearest cluster, $c_j$. In order to determine which cluster is closest to $x_i$, the Euclidean distance between $x_i$ and $c_j$ must be calculated, this is defined as the following:

$$D(x_i, c_j) = \sqrt{(c_j.RedValue - x_i.RedValue)^2 + (c_j.BlueValue - x_i.BlueValue)^2 + (c_j.GreenValue - x_i.GreenValue)^2}$$

Additionally, some very basic understanding of Nvidia GPU's block and thread structure may assist in understanding the basics of this algorithm. That information can be found here: https://www.nvidia.com/docs/IO/116711/sc11-cuda-c-basics.pdf.

**ALGORITHM 1:** Parallel K-means for GPUs

**for** *each cluster* $\in C$ **do**
  $cluster_j.RGB \leftarrow$ RGB value of some random point $x_i \in X$
**end**

initialize CUDA kernel
  $block\_per\_grid \leftarrow$ image_width
  $thread\_per\_block \leftarrow$ image_height
end CUDA kernel init

**while** *not converged* **do**
  call CUDA kernel
    currIndx = $thread.id * image\_width + block.id$
    $l\_currIndx \leftarrow$ value j where D($x\_currIndx$,$c_j$) is at a minimum
  CUDA kernel return to host

  Set all cluster meas to 0
  !!below here is the recalculating cluster means!!
  **for** *all pixels* $\in X$ **do**
    $cluster_i \leftarrow cluster_i + x_i$
    $total_i \leftarrow total_i + 1$
  **end**
  **for** *all clusters* $\in C$ **do**
    $cluster_j \leftarrow 1/(total_j * c_j)$
  **end**

  **if** *have any labels been reassigned from last iteration* **then**
    converged = false
  **else**
    converged = true
  **end**
**end**

It can be seen in algorithm 1 that was posed by Sirotković et al. [2012], that a single pixel can be assigned to its very own thread within the GPU. Then, the pixel can be addressed but taking the pixel's thread's id multiplied by the number of columns in the image and then adding the offset of the thread's block id. The algorithm itself is fairly straight forward. No special data structures are needed for implementation; although, I did use a struct to hold important information about each centroid.

## 3.1 My Implementation

**ALGORITHM 2:** Random Restart Parallel K-means for GPUs

**for** *Number of defined random restarts* **do**
  **for** *each cluster* $\in C$ **do**
    $cluster_j.RGB \leftarrow$ RGB value of some random point $x_i \in X$
  **end**

  initialize CUDA kernel
    $block\_per\_grid \leftarrow$ image_width
    $thread\_per\_block \leftarrow$ image_height
  end CUDA kernel init

  **while** *not converged* **do**
    call CUDA kernel
      currIndx = $thread.id * image\_width + block.id$
      $l\_currIndx \leftarrow$ value j where D($x\_currIndx$,$c_j$) is at a minimum
    CUDA kernel return to host

    Set all cluster meas to 0
    !!below here is the recalculating cluster means!!
    **for** *all pixels* $\in X$ **do**
      $cluster_i \leftarrow cluster_i + x_i$
      $total_i \leftarrow total_i + 1$
    **end**
    **for** *all clusters* $\in C$ **do**
      $cluster_j \leftarrow 1/(total_j * c_j)$
    **end**

    **if** *have any labels been reassigned from last iteration* **then**
      converged = false
    **else**
      converged = true
      **if** *this solution is the best among the random restarts* **then**
        $bestSolutionLabels \leftarrow currentSetOfLabels$
        $bestSolutionClusters \leftarrow currentSetOfClusters$
      **end**
    **end**
  **end**
**end**
Present best solution across random restarts

I did end up adding a small modification of my own to this algorithm in order to better handle the risk of local optimal solutions that this algorithm is susceptible to. I implemented a simple random restart modification, which we learned in class can assist with better end results of algorithms that rely on some form of random initialization of a solution and then proceed to improve that solution with respect to some heuristic. The exact number of random restarts is simply defined by the user. Finally, to compare the fitness of one solution to another, one simply examines the mean of the means of all the clusters. The smaller the mean of the means of the clusters, the smaller and tighter the clusters must be around their enclosing subjects. The tightest clusters possible for a given clustering problem is considered the optimal

solution. So, in an attempt to avoid most bad local optima, we simply take the tightest clustering configuration across all of the solution found by the random restarts.

## 4. TESTING AND VALIDATION

For validation, I ran this algorithm for k = 10, 20, 40, and 80 across 8 very different test images. For sanity checking, I used OpenCV's C++ library to present and save the found solutions for each picture. Additionally, I compared my algorithm's run time to the run times reported by Sirotković et al. [2012]. I will not present all of my results for all 4 k-values across all 8 pictures in this report for the sake of ease of reading. If you would like to view all of my results, simply look in the './Artificial-Intelligence-Self-Proposed-Midterm-Project/testImages/' directory. All of the results in the following section were ran under 40 random restarts.

### 4.1 Results

In the figures 1 - 4 below, it can be seen that k=40 is most certainly over-fitting all of the test images that I chose for this experiment. The best segmentation results (judge by eye for well defined border and clear objects) tend to hover around k=[8,13] for most of the pictures I chose to test on. I recommend opening the image files that you are interested in on a computer, pdf sizing limitations does not do the quality of the results justice.

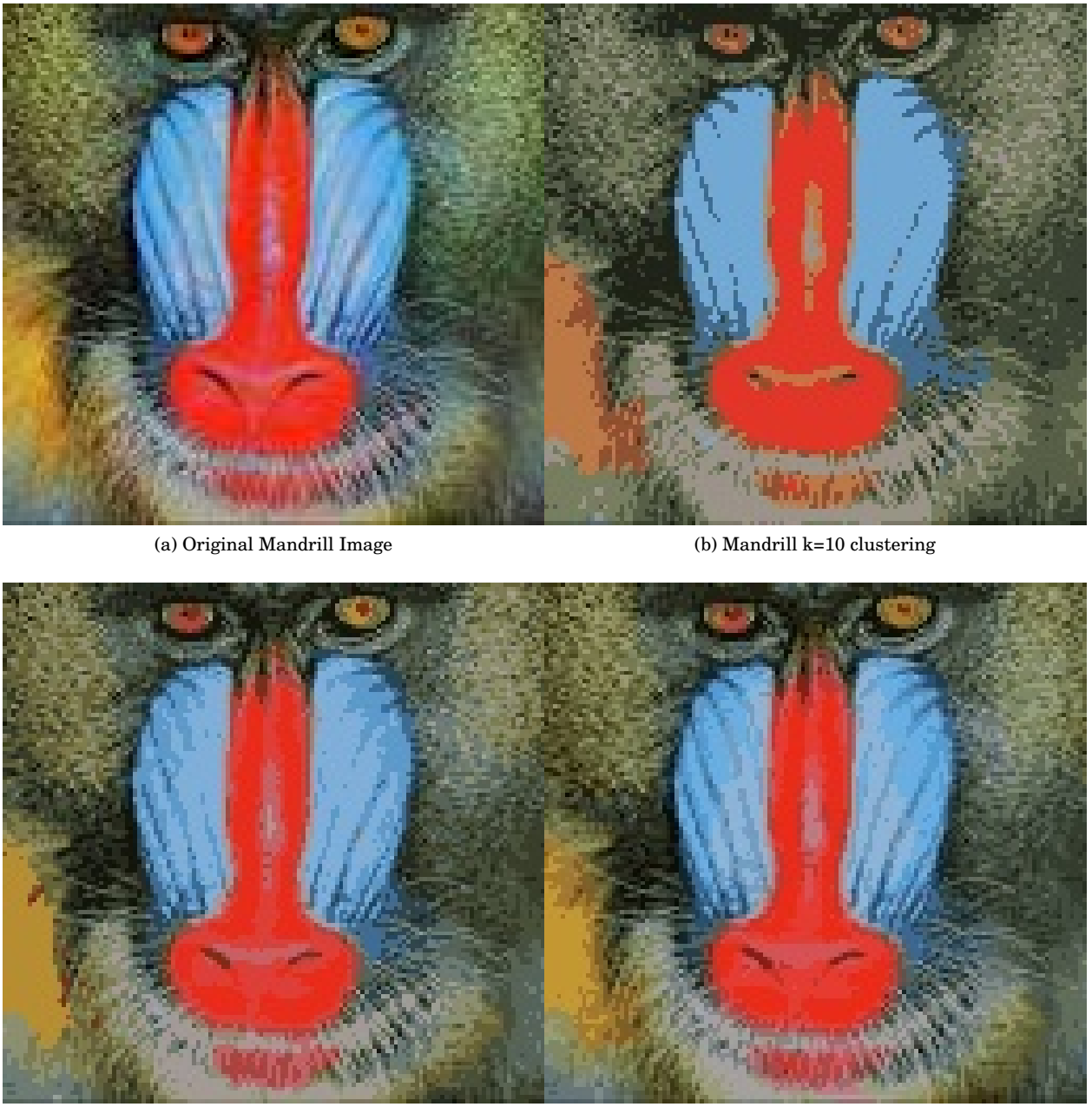
(a) Original Skittles Image

(b) Skittles k=10 clustering

(c) Skittles k=20 clustering

(d) Skittles k=40 clustering

Fig. 1: Skittles



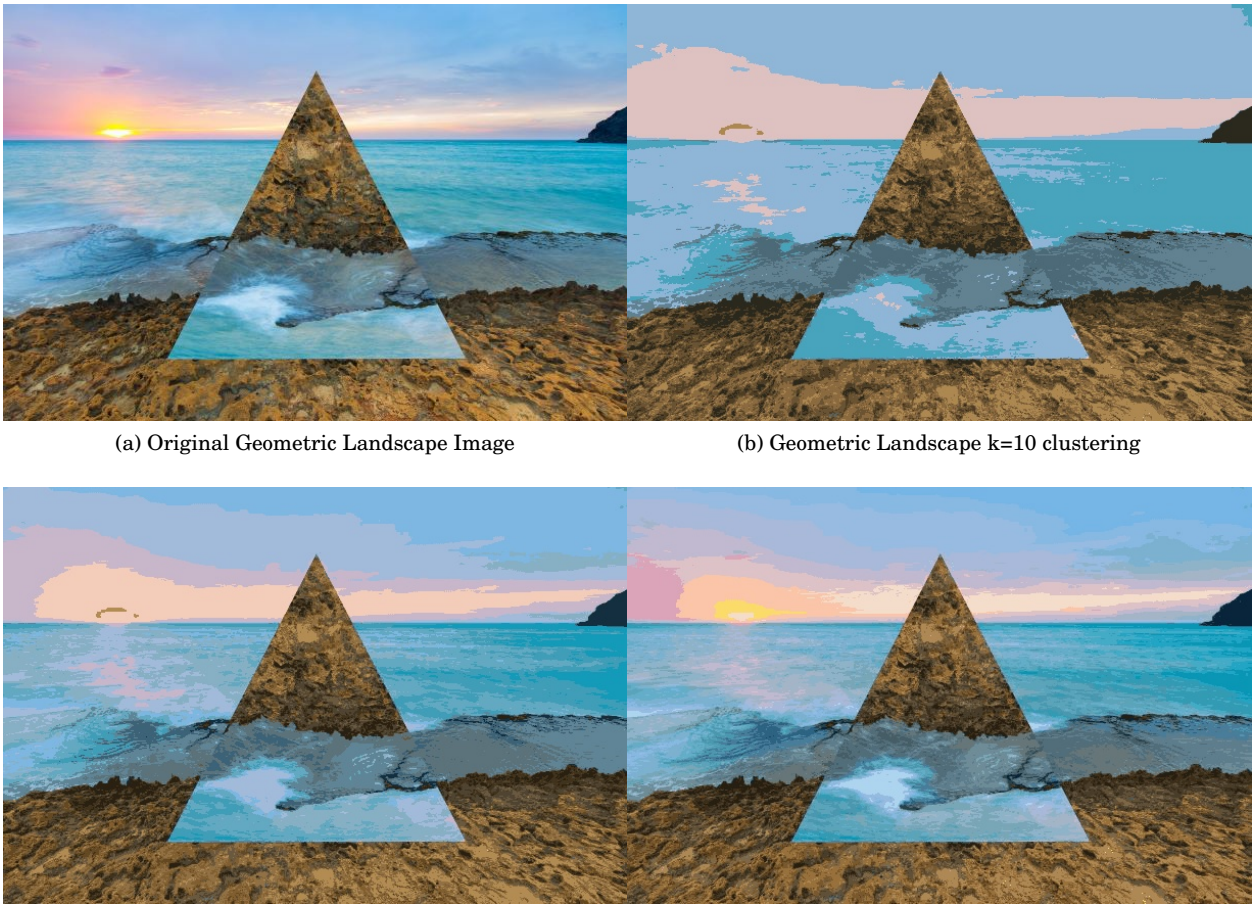
(a) Original Mandrill Image

(b) Mandrill k=10 clustering

(c) Mandrill k=20 clustering

(d) Mandrill k=40 clustering

Fig. 2: Mandrill



(a) Original Geometric Landscape Image

(b) Geometric Landscape k=10 clustering

(c) Geometric Landscape k=20 clustering

(d) Geometric Landscape k=40 clustering

Fig. 3: Geometric Landscape

(a) Original Hot Air Balloons Image

(b) Hot Air Balloons k=10 clustering

(c) Hot Air Balloons k=20 clustering

(d) Hot Air Balloons k=40 clustering

Fig. 4: Hot Air Balloons

In addition to these images, timing was also observed. It is important to note, that my code ran on an Nvidia GTX 1070 and the report from Sirotković et al. [2012] had results from running on a Nvidia 250 GTS. So, my GPU *should* blow theirs our of the water. Yet, my timing results did not even show as good of performance as the report says they achieved under their GPU. For k=10 on a 512x512 image, my code had an average convergence time of 0.185918s after all of the OpenCV operation were commented out. They achieved 0.04861449s. For k = 20 under the same setup, my code had an average convergence time of 0.249357s, while they showed 0.051679. I believe that this inconsistency could be caused either by some poor implementation decisions I made in my host code or the fact that they hard limited their converging loop to 20 iterations, but I could not narrow in on the exact cause. Regardless, my implementation runs much faster that the serial CPU implementation benchmarks reported by Sirotković et al. [2012]. So, the point still stands that GPU parallelized k-means is a superior implementation of the k-means algorithm compared to its trivial implementation.

## 5. CONCLUSION

In conclusion, GPU parallelization of the k-means algorithm is a simple and effective way to greatly improve run-time cost when compared to the trivial case. Although I could not match the time results achieve by the original implementers of this code, I was still able to achieve run times that are drastically improved over the serial i3 CPU benchmarks that were reported by Sirotković et al. [2012]. Today, it is now a

known fact that machine learning algorithms thrive on SIMD architecture because of the common independent nature of the subject data and repetitiveness of operations.

NOTE: SORRY! I couldn't find where in the ACM latex template where I could change the bibliography to 12pt font!!

Indirectly used sources to add to bib: [Wikipedia 2018b] [Wikipedia 2018a]

REFERENCES

Jadran Sirotković, Hrvoje Dujmić, and Vladan Papić. 2012. K-Means Image Segmentation on Massively Parallel GPU Architecture. In *Proceedings of the 35th International Convention MIPRO*. IEEE, 489–494.

Andrea Trevino. 2016. Introduction to K-means Clustering. (December 2016). Retrieved October 10, 2018 from https://www.datascience.com/blog/k-means-clustering

Wikipedia. 2018a. Image Segmentation. (September 2018). Retrieved October 10, 2018 from https://en.wikipedia.org/wiki/Image_segmentation

Wikipedia. 2018b. k-means clustering. (October 2018). Retrieved October 10, 2018 from https://en.wikipedia.org/wiki/K-means_clustering