ADVANCED COMPUTER ARCHITECTURE

PROJECT REPORT

Color-based Image Segmentation

using Parallel K-means Clustering

*Author:*

Gabriele MIRANDO

A.Y. 2017/2018

**Abstract**

The objective of the project was to develop a C language program for color-based image segmentation using the standard k-means clustering algorithm, and to speed up the execution by implementing a parallel version of the program able to run on multi-core CPUs. This report contains a complete account of how the work has been organized and what results have been obtained. It starts with a detailed analysis of how the standard k-means algorithm can be exploited to achieve color-based image segmentation through the clustering of the pixels of the image. An a-priori study of the available parallelism is then conducted, taking into account the results obtained from the profiling of the serial code. The algorithm has been parallelized using OpenMP. A complete description of the parallel implementation is provided. Finally, multiple test cases for the program are taken into consideration and some observations about performance are deduced from the comparison of the execution times obtained on machines with an increasing number of cores.

Table of Contents

Analysis of the Serial Algorithm 4

Implementation Walkthrough 5

Serial Algorithm Results 9

Study of the Available Parallelism 10

Serial Code Profiling 11

Speedup Estimation 11

Description of the Parallel Implementation 12

OpenMP Implementation 13

Performance Analysis 14

Speedup on a Local Machine 14

Speedup on Google Cloud Platform 14

# Analysis of the Serial Algorithm

Image segmentation is the process of partitioning a digital image into its constituent objects. Given a digital image, performing a segmentation means to identify the main elements that make up the scene. Achieving a good quality segmentation is one of the most difficult and challenging tasks in digital image analysis, but it also represents the first step towards an effective detection of the objects present in the image. The more accurate is the segmentation phase, the more likely is the image recognition process is to succeed. For the purpose of the project, a simple color-based segmentation technique has been considered.

The color-based approach relies on the fact that in a digital image sub-regions of pixels can be identified on the basis of the their color. The partitioning of the pixels can be achieved by the means of a clustering algorithm. Clustering, in fact, is by definition the task of organizing a given set of objects into groups, or more precisely into clusters, in such a way that objects in the same cluster are more similar to each other than to those in other clusters. Therefore given an initial image and an integer number K, using a clustering algorithm it is possible to divide the image pixels into K groups, such that pixels in the same group are similar in terms of color.

The fact that is extremely easy to implement, but is also computationally very efficient, made the algorithm proposed by Stuart P. Lloyd in 1957 the most popular clustering technique used in scientific and industrial applications. This technique is known as k-means clustering algorithm. Although multiple variants of k-means have been later developed (Forgy, MacQueen, Hartigan-Wong), Lloyd’s version is considered the standard version of the algorithm and it is the one that has been used for achieving color-based segmentation in this project.

The aim of the program is, given an initial digital image saved in one of most the common format (JPEG, PNG, BMP, GIF…) and an integer number K, to obtain a segmented image where K different regions pixels are highlighted and easily discernible. The execution of the program is organized into three phases: the reading of the RGB color values of the pixels of the starting image, the segmentation via k-means clustering of those values and then writing of the obtained results to a new image, that will represent the final outcome of the segmentation. The activity focused on implementing the C language code for the segmentation using k-means, while an open source library available on Github has been adopted for convenience to handle the image I/O operations.

## Implementation Walkthrough

In its most general definition, the standard k-means algorithm can be used to partition a set of objects, described by a series of measurable features, into a predefined number of clusters. When used for color-based image segmentation, the objects are the pixels of the image and the measurable features are their RGB values. In the program, the color values obtained by scanning the input image are stored in a matrix allocated in memory and constitute the initial dataset on which to perform the segmentation. Each row of the matrix contains the 0 to 255 RGB values of each pixel of the image. Given a digital image of width W and height H, the initial dataset is a matrix made up by N = W x H rows and three columns:

where are respectively the red, green and blue components of the i-th pixel of the image. The segmentation of the image pixels using k-means clustering algorithm is achieved by the program by following these steps:

1. **Initialization of the clusters centers**. K pixels are randomly picked from the dataset and set as initial cluster centers. A matrix of K rows is allocated in memory to store the values of the clusters centers:

where are respectively the red, green and blue real values of the i-th cluster center. Each cluster center is a vector of three components that need to be initialized with the values of a random pixel.

1. **Assignment of each pixel to the nearest cluster**. For each of the pixels of the dataset, the squared Euclidean distance from all the cluster centers is computed. The cluster center for which the value of the squared Euclidean distance is the lowest is found and the pixel is assigned to that cluster. The squared Euclidean distance between one pixel and a cluster center is the sum of the squared differences of the values of the color components and measures how much a given pixel and the pixels belonging to the cluster are similar in terms of color. To keep track of the cluster to which each pixel belongs at each iteration of the algorithm, an array of size N has been stored in memory:

where is an integer value from 0 to K-1 that indicates the index of the cluster to which the i-th pixel currently belongs. Another vector of N elements is used to store the squared Euclidean distances of each pixel to the center of the cluster it belongs:

where is a real value that contains the squared Euclidean distance of the i-th pixel to its cluster center. Even if not strictly necessary, the *dists* vector can be very useful in speed up some computations, like the one of the SSE at step 5. Both the *labels* and *dists* vectors are updated at this step.

1. **Update of the cluster centers**.For each cluster, the value of the center is recomputed by calculating the mean of all the pixels belonging to the cluster. In the case of a cluster being empty, a good practice is to set its new center to the pixel of the dataset with the maximum distance from the center of the cluster to which the pixel belongs. The *centers* vector is updated at this step.
2. **Test for completion**. The algorithm is returned to step 2 until the cluster assignments do not change (this is when the algorithm converges to a solution) or until a predefined maximum number of iterations is reached. If none of the pixels has changed the cluster to which they belong or a the number of maximum iterations is reached, the clustering is completed, but an additional simple steps is required to obtain the segmented image.
3. **Compute the SSE**. The SSE is easily computed by summing all the elements of the *dists* vector. The objective of the k-means algorithm is to find the clusters composition that minimize the Sum of Squared Errors (SSE), i.e. the sum of the squared Euclidean distances from each pixel in the dataset to the center of the cluster to which the pixel belongs. K-means is an heuristic procedure that attempts to find iteratively and in the fastest way possible an approximation of the optimal solution for the problem of the minimization of the SSE. The algorithm allows to obtain just a local minimum, not the global minimum, and the cluster composition obtained at convergence depends on the initialization of the centers at the first step. Although the computation of the SSE is essential, comparing the SSE among different executions of the algorithm can be useful to check the goodness of the result. Selecting the initial centers at random among the objects of the dataset allows to achieve acceptable results in the case of the application considered for the project, but a more sophisticated initialization technique or multiple repetitions of the algorithm may be required for other applications, especially in the field of data analysis.
4. **Update image data.** The RGB values of each pixel of the image are replaced with those of the cluster center to which the pixel belongs. In this last step the *data* matrix is updated and will be used as data source of the final segmented image.

A visual representation of the algorithm for color-based segmentation using k-means clustering is shown by the flowchart of Figure 1. The core of the program, i.e. the part that does the majority of the work, is the loop where the assignment of the pixels to the clusters and the update of the centers take place.

**Initialize clusters centers**, randomly picking K pixels from the initial image

**Assign each pixel to the closest cluster**, by choosing the cluster for which the squared Euclidean distance is the lowest

**Update clusters centers**, computing the mean of all the pixels belonging to the cluster

Any pixel changed cluster?

yes

no

**Compute SSE,** summing the squared Euclidean distances of each pixel to its closest cluster center

**Update image**, replacing each pixel RGB values with the ones of center of the cluster to which the pixel belongs

**Figure 1.** The flowchart for color-based segmentation using k-means clustering algorithm

## Serial Algorithm Results

The main test case of the program has been a 640 × 360 JPEG image of a horse. The test image has been segmented with increasing values of K. Incrementing the number of cluster meant to increase the number of computations of k-means but also the number of iterations that took the algorithm to converge, which traduced in higher execution times.

Regarding the final segmented image, Figure 2 shows that images with different levels of accuracy were obtained and that the choice of K was crucial for a good quality segmentation. Four colors were immediately distinguishable in the original image and after performing the segmentation with K = 4, the shape of the horse was clearly identifiable.

|  |  |
| --- | --- |
| **(a)** The original JPEG image | **(b)** Color-based segmentation with K = 4 |
| **(c)** Color-based segmentation with K = 8 | **(d)** Color-based segmentation with K = 16 |
| **(e)** Color-based segmentation with K = 32 | **(f)** Color-based segmentation with K = 64 |
| **(g)** Color-based segmentation with K = 128 | **(h)** Color-based segmentation with K = 256 |

**Figure 2.** The horse image segmented using different values of K

When increasing the value of K, the SSE returned by the k-means algorithm was obviously decreasing and the segmented image started to resemble the original image. Setting K = 256 basically allowed to obtain a 256-color palette version of the initial picture.

The color-based technique adopted for the project is naive and truly effective only in particular circumstances, and it’s rarely adopted when the goal is to achieve an high quality segmentation. However, the real purpose of the activity was to implement, parallelize and assess the performance speedup of the k-means algorithm, which has been used for the clustering of the pixels. Color-based image segmentation has just been chosen among the many applications of the k-means algorithm for its practicality and originality.

# Study of the Available Parallelism

For an effective parallelization of the serial code, the methodology adopted was to assess what were the sections of the algorithm were the program was spending the most time during the execution and evaluate if those sections could actually be parallelized.

## Serial Code Profiling

The first step taken towards a parallel version of the program was the profiling of serial of the serial code using the GNU profiler *gprof.* Thanks to the Call Graph profiling module of *gprof* it was possible to get the percentage of execution times taken by each function of program. The code was organized in the code in such a way that each function corresponds to one of the steps of the k-means segmentation algorithm. The profiling was conducted for the program with an increasing number of clusters and results are reported in Table 1. The functions that took care of the image I/O operations were not considered, but the profiling and the activity of parallelization focused only on the k-means segmentation.

What is immediately noticeable from Table 1 is that the most of work is done by the program in the main loop of the k-means algorithm and in particular when each pixel need to be assigned to its closest cluster. When increasing the number of cluster K, this function became the one that required by far the majority of computations, reaching 99% of the execution time of the entire segmentation process. Just the parallelization of the *assign\_pixels* could guarantee an acceptable speed-up.

## Speedup Estimation

Each function need to be considered on his own and the feasibility of the realization of parallel version discussed in details:

* **init\_random:** clusters center are initialized by selecting at random K pixels from the image dataset. The *rand* function in C is not thread safe. Parallelizing this function may require extra effort, but without getting having an impact of the final speedup.
* **assign\_pixels:** the assignment of each pixel to the closest cluster center is what is known an embarrassingly parallel problem. The computation of the closest cluster of each pixel can be done separately on different threads without the need of synchronization. The *data, centers, labels* and *dists* structures will be shared, but without the need of worrying for race conditions or any kind of conflicts because each thread will operate on different memory addresses.
* **update\_centers:** updating cluster centers requires to compute the mean of the pixels values belonging to each cluster. To parallelize this function the *centers* matrix need to be opportunely synchronized or accessing and updating from different threads can lead to race conditions.
* **compute\_sse:** computing the SSE can be done by summing all the values of the *dists.* This implies a loop-carried dependence that need to be considered, even the parallelization of this function won’t make a difference on the final speedup.
* **update\_data:** updating the image pixels values, by replacing them with those of the cluster center to which each pixel belongs, can be easily done in parallel without the need of synchronization.

The key to obtain a relevant speedup is to parallelize the *assign\_pixels* function.

It is possible to use Amdahl’s Law to make an estimation of the possible speedup achievable just by parallelizing the step of the algorithm of the pixels assignment to their closest cluster.

The results of Amdahl’s Law have been used to realize the graph of Figure 3. The maximum speedup achievable is of course higher when the number of clusters K is higher, since the assign\_pixels function represents a bigger portion of the overall execution time.

# Description of the Parallel Implementation

By the study of the available parallelism of the k-means clustering algorithm, it resulted that a remarkable speedup can be obtained by spreading the search for the closest cluster of every pixel at each iteration of the algorithm over multiple threads. K-means is a good example of algorithm where an increase in performance can be achieved thank to data parallelism. Each thread perform the same work on different parts of some shared structures. The same job is performed in parallel on different data across multiple processors.

## OpenMP Implementation

The program has been parallelized using *OpenMP*. *OpenMP* is application programming interface specifically developed for multiprocessing programming using C language. It offers a set of compiler directives, library routines and environmental variables that can be used to make a C source code to run on multicores CPUs. With the use of *pragmas* it was possible to implement a parallel version of the k-means segmentation algorithm with a reduced effort.

The first step was to parallelize the assignment of each pixel to the closest cluster. This was done using the *parallel for* work-sharing construct.

*#pragma omp parallel for schedule(static) private(…)*

*for (px = 0; px < n\_px; px++) {*

*…*

*}*

The work that is done inside the for loop is shared across the threads available to the program. The scheduling preference is set to static so that an equal portion of work is assigned to each thread in advance. Supposing that we are considering a machine with 4 CPU threads available and an image with 40000 pixels. In this loop, for each of the 40000 pixels the closest cluster need to be computed. Using the *parallel for* with static scheduling will assign in advance the first 10000 pixels to the first thread, the second 10000 pixels to second threads and so on. This approach is much more efficient than dynamic scheduling in this this case, because the work distribution is well balanced. All the threads will complete their job almost at the same time because the computations they need to do to find the closest cluster of each pixel are almost always the same. They need to compute the squared Euclidean distance between the pixel and every cluster center and find the minimum. A dynamic approach where the portions of work are assigned at run-time when the threads have completed their job would be a waste of time due to the communication overhead. Using this simple pragma inside the *assign\_pixels* functions an already remarkable speedup can be obtained. The exact same approach has been utilized in order to parallelize the update of the final image.

Parallelizing the *update\_centers* has revealed to be a little bit more complicated. To compute the new clusters center, i.e. the mean of the pixels belonging to each cluster, a loop has been used to scan for all the pixels. For each pixel the program has to check the cluster of each pixel and update the centers matrix, containing now the partial sums of the pixels values of each cluster, and a counts vector, keeping track of the number of pixels belonging to each cluster. Updating the centers matrix or the counts vector from multiple threads may lead to race conditions and unexpected results of the program. A synchronization technique need to be adopted. Since using a critical section would compromise performance, a reduction clause, to protect the update operations of the centers and the cluster counters, has been chosen instead.

*#pragma omp parallel for private(…) reduction(+:centers[..],counts[..])*

*for (px = 0; px < n\_px; px++) {*

*…*

*}*

Since the version 4.5 of OpenMP it is possible to perform reductions also on vectors. The reduction clause has been used in the same way for computing the SSE.

# Performance Analysis

In order to evaluate the efficiency of the parallel implementation realized using OpenMP, the program has been tested on different multi-cores machines and the speedup in respect to the serial version has been computed. The program has been executed by varying the number of clusters K to use for image segmentation.

## Speedup on a Local Machine

## Speedup on Google Cloud Platform