EE 451- Parallel and Distributed Computation Final Project Report

Achieving Image Segmentation Speedup Through Parallelizing Quick Shift Algorithm

Sufian Mohammad
Taryn Sanders

University of Southern California (USC) - Los Angeles, CA

Spring 2024 - Professor Viktor Prasanna

Introduction

Image processing has become a necessary discussion in various emerging industries such as computer vision, medical imaging, autonomous driving, and artificial intelligence [1]. With its aid in image analysis through improved visual quality and ability to extract useful information, this method and its encompassing techniques have been at the forefront of extensive research [2]. Of the techniques, one of the most critical is image segmentation.

This [image segmentation] particular technique, which involves partitioning an image into various regions based on the properties of pixels, not only reduces the complexities of the image for a machine [3], but also determines the quality of the final output of analysis [2].

Despite image segmentation being a component of the larger method of image processing, the aforementioned has numerous approaches. Of those implemented, some of the most common include [4]:

1. Thresholding: This technique involves selecting a threshold value to separate objects from the background based on pixel intensity. It is a simple and efficient method, particularly when there is a distinct contrast between foreground and background. However, it may struggle in cases where the intensity values overlap or when the lighting conditions vary across the image.

2. Region-based Segmentation: These methods group pixels into regions based on similarities in color, texture, or other features. One popular algorithm is the Watershed algorithm, which treats pixel intensities as a topographic surface and separates regions based on watershed lines. Region-based techniques are effective in scenarios where objects have varying intensities or when there are discontinuities in the image.

3. Edge Detection: This technique identifies boundaries by detecting sharp intensity transitions or gradients in the image. Popular edge detection algorithms include the Canny edge detector and the Sobel operator. Edge-based methods can accurately segment objects with well-defined boundaries but may struggle with noisy or ambiguous images.

4. Clustering: Clustering algorithms, such as K-means or Mean-Shift clustering, group pixels into clusters based on their similarity in feature space. It is a versatile technique that can handle complex images with multiple objects. Clustering-based approaches are effective when the number of segments is not known in advance and can adapt to variations in object color, shape, or texture.

5. Deep Learning-Based Segmentation: With the advent of convolutional neural networks (CNNs), deep learning has revolutionized image segmentation. Techniques like U-Net, Mask R-CNN, and Fully Convolutional Networks (FCNs) have achieved remarkable performance by leveraging the power of

neural networks for pixel-level classification. Deep learning-based segmentation methods can handle complex images, exhibit robustness against noise, and learn complex patterns and relationships.

In this report, the primary focus will be region-based segmentation by introducing the Quick Shift algorithm for image segmentation. It is based on an approximation of Mean Shift Clustering. In its entirety, Quick Shift is a hierarchical clustering technique. It uses a non-parametric method which aims to identify homogeneous regions in an image based on similarities in color and spatial proximity. It starts with representing each pixel in the image as a data point in a feature space, consisting of color values and spatial coordinates.

Quick Shift operates iteratively by merging clusters based on their similarity. It computes a density estimate at each pixel location, which represents the likelihood of finding similar pixels in its neighborhood. This density estimate is calculated using a kernel function, such as the Gaussian kernel, and takes into account both color and spatial proximity.

In each iteration, the algorithm identifies the neighboring clusters with the highest density and merges them together. In essence, Quick Shift arranges all of the data points into a tree, where parents in the tree are the nearest neighbors in the feature space, which increases the estimate of the density. This process continues until convergence, which occurs when the clusters stabilize and no further merges are possible. The resulting segmentation is obtained by assigning each pixel to the cluster it belongs to after convergence.

In this project, we aim to achieve the following tasks:

1. Implement a serial version of Quick Shift using C programming.
2. Implement a parallel version of Quick Shift using CUDA programming and shared memory.
3. Evaluate speedup on a GPU platform using different images of various complexities with our serial baseline.

Algorithm

Given N pixels, $x_1, x_2, \ldots, x_N$, we start by computing a Parzen density estimate around each point:

$$P(x) = \frac{1}{2\pi\sigma^2 N} \sum_{i=1}^{N} e^{\frac{-\|x-x_i\|^2}{2\sigma^2}} \quad (1)$$

Once the density estimate has been computed, Quick Shift connects each point to the nearest point in the feature space, which has a higher density estimate. Each connection has a recorded distance associated with it, and the set of pixel connections forms a tree, where the root of the tree is the point with the highest density estimate.

Algorithm 1: Parallel Quick Shift Based Image Segmentation Algorithm

```
Input  : data collection $\chi$
         broken distance $\delta$
         window size $r$
Output: Collection of tree contains all pixels
         with the property distance
function Quickshift $(\chi, \delta, r)$;
for $x_i \in \chi$ do
    $P(x_i) = 0$
    for $x_j \in$ pixels less than $r$ away do
        $P(x_i) \mathrel{+}= e^{-\frac{\|x_i - x_j\|^2}{2\times(r/3)^2}}$
    end
end
for $x_i \in \chi$ do
    for $x_j \in$ pixels less than $\delta$ away do
        if $P(x_j) \le P(x_i)$ and $\|x_i - x_j\|^2$ is smallest then
            $distance(x_i) = \|x_i - x_j\|^2$
            $parent(x_i) = x_j$
        end
    end
end
```

To obtain a segmentation from the tree of links, we choose a threshold, say τ, and break all the links in the tree where the intra-pixel distance is greater than τ. The pixels in each segment of the resulting disconnected tree form a segment.

Context

We first provide some definitions.

Mode seeking: Start by computing the Parzen density estimate k(x), where k(x) can be Gaussian or another window. These algorithms require a numerical scheme to evolve the trajectories, a halting rule to decide when to stop the evolution, and a clustering rule to merge the trajectory end points.

Mean Shift Algorithm: This mode-seeking, non-parametric clustering algorithm utilizes the idea of associating each data point to a mode of the underlying probability density function. By approaching the mapping in this way, the cluster's structure is arbitrary, and the number of clusters needed doesn't need to be known in advance. This algorithm is essentially a gradient ascent algorithm in that the gradient may not be defined unless the data space has additional structure. In practice, this algorithm takes various data points and guides them towards the mode, approximately following the gradient. The time complexity of this algorithm is $O(dN^2T)$, where d is the dimensionality of the data space and T is the number of iterations[4].

Medoid-Shift Algorithm: While this algorithm has some similarities to the aforementioned Mean Shift, this particular mode-seeking algorithm extends easily to general metric spaces. This allows the algorithm to be used on curved spaces without additional steps. This algorithm is also non-iterative and therefore doesn't require stopping heuristics. A disadvantage is that this algorithm fails to cluster data points belonging to the same mode, which results in over-fragmentation. Another disadvantage is that the computational complexity of this algorithm is between $O(dN^2 + N^3)$ [4]. In practice, this algorithm approximates the trajectories of the Mean Shift algorithm connecting data points. Essentially, what this means is that Medoid shifts are constrained to connect points in the area assigned to the data points. This unfortunately means that where the data is sparse, disconnections may occur, but this can be alleviated by iterating the procedure. Medoid-Shift is considerably faster than Mean Shift. The computational complexity of Euclidean medoid shift is $O(dN^2)$ with a small constant. When it is generalized to non-Euclidean distances, kernel methods are used, which shows that the complexity is bounded by the effective dimensionality of the kernel space [4]. The weak point of this algorithm is its inability to consistently identify all the modes of the density. This concern was addressed implicitly by reiterating medoid shift on a simplified dataset, but it comprises the non-iterative nature of the algorithm and changes the underlying density function.

A modification of Mean Shift where the trajectories are constrained to pass through the points is discussed. The advantages include: only one step has to be computed for each point, there is no need for stopping/merging heuristics because the conditions are exactly met, and the data space X may be non-Euclidean. Drawbacks include the speed, with a basic implementation being $O(dN^2 + N^3)$ [4]. Even using the fastest matrix multiplication algorithm would only improve speed to $O(dN^{2.38})$ with a large hidden constant, which is impractical. Therefore, the asymptotic estimate is $O(dN^2 + N^{2.38})$ [4].

Quick Shift Algorithm: This algorithm explicitly trades off under-fragmentation and over-fragmentation. The algorithm operates in non-Euclidean spaces in a straightforward manner. Quick Shift simply moves each point to the nearest neighbor for which there is an increment in density. It has four advantages: simplicity, speed, which is $O(dN^2)$ with a small constant, generality, and a tuning parameter to trade off over-fragmentation and under-fragmentation of the modes [4]. This algorithm connects all the points into a single tree, modes are then recovered by breaking the branches that are longer than the threshold τ as previously discussed. One thing that is specifically unique is that the gradient is maximized in a neighborhood of each point defined by the choice of the threshold parameter. This algorithm returns the solution for the possible thresholds at once, making the model selection more efficient.

Parallelization Strategy

Because Quick Shift operates on each pixel of an image and the computation of density estimates that takes place at each pixel is independent of its neighboring pixels' computations, it makes sense to implement Quick Shift on a parallel architecture.

We propose to use CUDA programming to copy the image to the GPU unit and break down the computation of the density of a pixel and its surrounding neighbors into multiple thread blocks. We propose to initialize a grid equal to the image dimensions and assign each pixel its own block of threads, where the number of threads is equal to a certain size, up to the size of the window, depending on the size of the chunk of neighboring pixels (as small as 1 pixel per thread) we decide to allocate to each thread.

We also need to address memory access times. Given that each pixel needs to access $(6\sigma + 1)^2 - 1$ neighbors, memory latency increases quadratically with σ. From the lectures, we know that each GPU has global memory. However, global memory is slow and takes hundreds of cycles to access. Additionally, each streaming multiprocessor has its own shared memory for all threads in a block to access. We propose to load a chunk of surrounding neighboring pixels into shared memory to promote data reuse. When a pixel in a block tries to compute similarity with a pixel outside of that block, the memory access can be cached to shared memory.

Hypothesis

Thus, our hypothesis is as follows:

By parallelizing the density calculation, the computational time and complexity can be improved by using shared memory provided by streaming multiprocessors and exploiting data reuse.

Experimental Setup

Data Preparation

For this project, four datasets will be used for evaluation.

Image ID	Image Visual	Size
1		512 x 512
2		612 x 408
3		300 x 168

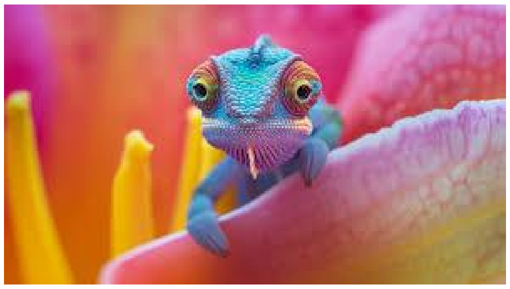
4		300 x 168

Table 1. Parameter of Datasets

This dataset provides us with diverse images ranging in size and complexity. The first image, while being relatively detailed, has defined color blocks which should allow for easy placement of superpixels. The second image introduces image complexity which is shown in the differentiation of the sky and the tops of the trees, thus requiring more superpixels. The third image tests the algorithm's ability to accurately place the superpixels. This image is straightforward in terms of color blocking, but because there are so many colors, the number of superpixels is large and could potentially impact the image output. The last image combines both image complexity, with the overlapping color detail on the leaf, and the overall number of colors.

Evaluating Performance

To evaluate performance, we are looking at two things: image quality and execution time. To deem the algorithmic speed-up successful, the image quality must be comparable to not only the original image, but the serial baseline as well. To measure this we will look at overall image identification, whether you can clearly make out what the image is portraying, and image complexity replication, where numerous colors overlap, how accurate was the Quick Shift algorithm at determining the proper superpixel.

The second evaluation metric we will use is computation time. Despite the Quick Shift algorithm being one of the fastest in terms of image segmentation, the success of its parallelization will be heavily dependent on the overall execution time. While the exact margin of speedup remains unknown, we expect to see a decrease in the overall time and a slight increase in the number of superpixels used.

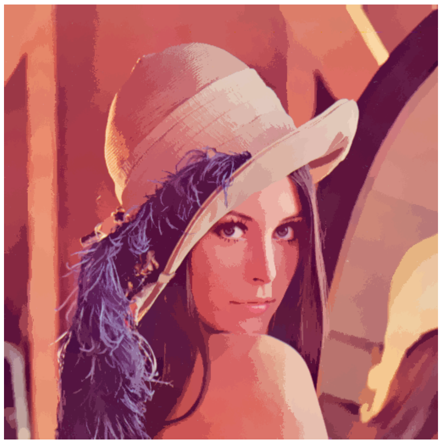
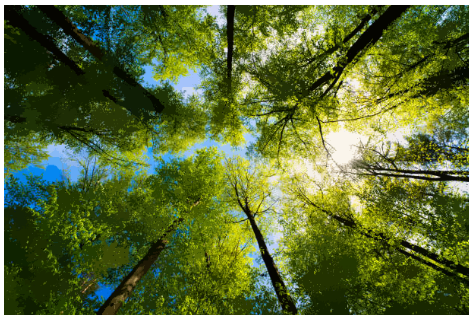
Serial Implementation

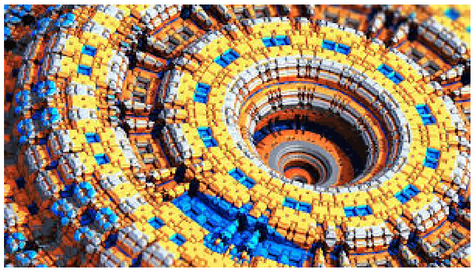
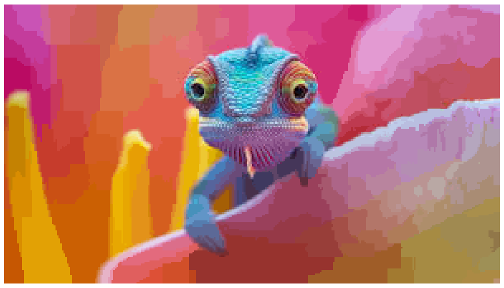
To achieve the baseline data, we will be using a C++ programming version of the Quick Shift algorithm that is run in Visual Studio Code. VS Code utilizes the hardware components of your local device and therefore the parameters cannot be generalized. With this in mind, the outputs may also slightly vary due to the hardware components across devices.

Results and Analysis

Serial Implementation Results

As can be seen from the data the expectations detailed above were relatively spot on: the larger the image, the slower the computation time; the more complexity the more superpixels. The interesting thing this baseline revealed was the data produced for Image #4. While initially we thought this image was the most complex, this image not only took the shortest amount of time, but used the fewest number of superpixels. At first glance this image looks almost identical to the original, however looking more closely, specifically at the leaf, we see that the accuracy was lost. Despite numerous possibilities as to why this occurred, the most likely is that the superpixels within that region were too close to each other and the weights of the leaf detailing couldn't be properly assigned to the correct superpixel, leading to the blurry output.

Image ID	Output Image	Image Size	# of Superpixels	Computation Time (ms)
1		512 x 512	6,276	10224.0660
2		612 x 408	68,892	10153.0800

3		300 x 168	35,554	2116.3860
4		300 x 168	3,402	1807.1820

Parallelization Output

Unfortunately, we were unable to create a working parallelization algorithm to compare to the baseline. While we tried numerous methods such as modifying open source algorithms, creating our own, etc, the results always ended in one superpixel with an output image of all one color.

Another problem we encountered when attempting to create the parallelization algorithm was the lack of adequate technology. Due to our local computers not having the proper GPU hardware to accommodate the parallelization, we relied on CARC. This unfortunately caused a series of issues we were unable to resolve during the duration of this project.

Nonetheless, we still expect the parallelization of the Quick Shift algorithm to be quicker than the serialized baseline. In the future we hope to gain access to adequate hardware, which will allow us to test and debug the parallelization algorithm to produce the results in which we can compare.

Conclusion

Image processing is crucial in emerging fields like computer vision, medical imaging, and artificial intelligence. Image segmentation, a key technique, involves partitioning images into regions based on pixel properties, reducing complexity and enhancing analysis quality.

The report primarily focuses on region-based segmentation, introducing the Quick Shift algorithm. Quick Shift is a hierarchical clustering method that identifies homogeneous image regions based on color and spatial proximity similarities. It operates iteratively by merging clusters with high similarity, resulting in a segmentation obtained by breaking branches of the connection tree at a given threshold.

To accelerate Quick Shift computation, a parallel version using CUDA programming and shared memory is proposed. This approach aims to exploit GPU architecture, particularly benefiting from shared memory for data reuse and reducing memory access latency.

Four datasets with diverse image sizes and complexities are used for evaluation. Performance is measured based on image quality and execution time. The serial baseline implementation in C++ revealed trends: larger images and more complexity lead to longer computation times and higher superpixel counts, with potential accuracy issues.

Unfortunately, a working parallelization algorithm was not achieved due to hardware limitations and technical challenges, resulting in uniform output images. However, it's anticipated that parallelization will improve computation time, pending access to suitable hardware.

Source Code

EE 451 Final Project - Quick Shift

References

[1] D. T. LLC, “What Is Image Processing : Overview, Applications, Benefits,” LinkedIn, https://www.linkedin.com/pulse/what-image-processing-overview-applications-benefits/ (accessed Apr. 28, 2024).

[2] H. D. Cheng, X. H. Jiang, Y. Sun, and J. Wang, “Color image segmentation: advances and prospects,” *Pattern Recognition*, vol. 34, no. 12, pp. 2259–2281, Dec. 2001. doi:10.1016/s0031-3203(00)00149-7

[3] F. Sultana, A. Sufian, and P. Dutta, “Evolution of Image Segmentation using Deep Convolutional Neural Network: A Survey,” *Knowledge-Based Systems*, vol. 201–202, p. 106062, Aug. 2020. doi:10.1016/j.knosys.2020.106062

[4] H. Bandyopadhyay, “An Introduction to Image Segmentation: Deep Learning vs Traditional,” V7 Labs, https://www.v7labs.com/blog/image-segmentation-guide (accessed Apr. 28, 2024).

[5] A. Vedaldi and S. Soatto, “Quick Shift and Kernel Methods for Mode Seeking,” *Lecture Notes in Computer Science*, pp. 705–718, 2008. doi:10.1007/978-3-540-88693-8_52

[6] B. Fulkerson and S. Soatto, “Really Quick Shift: Image Segmentation on a GPU,” *Trends and Topics in Computer Vision*, pp. 350–358, 2010. doi:10.1007/978-3-642-35740-4_27

[7] Ryan Potter, “Why Image Segmentation is Needed: Image Segmentation Techniques”, Medium, https://becominghuman.ai/why-image-segmentation-is-needed-image-segmentation-techniques-ee52b92e651a (accessed Apr. 28, 2024)

[8] Mrinal Tyagi, “Image Segmentation Explained”, Builtin, https://builtin.com/machine-learning/image-segmentation (accessed Apr. 28, 2024)

[9] Brain Fulkerson and Andrea Vedaldi, “Quick Shift Image Segmentation”, VLFeat.org, https://www.vlfeat.org/api/quickshift.html#:~:text=Quick%20shift%20%5B29%5D%20is%20a,a%20basis%20for%20further%20processing. (accessed Apr. 28, 2024)

[10] D. Comaniciu, P. Meer: Mean Shift: A Robust Approach toward Feature Space Analysis, IEEE Trans. Pattern Analysis Machine Intel., Vol. 24(5), 603-619, 2002

[11]Yaser Sheikh, Erum Khan, Takeo Kanade, "Mode-seeking via Medoidshifts", IEEE International Conference on Computer Vision, 2007.

[12]A. Vedaldi and S. Soatto, "Quick Shift and Kernel Methods for Mode Seeking," in Proceedings of the European Conference on Computer Vision (ECCV), 2008