Jonathan Song

Haoran Fang

CSCI 4830-012, Concurrent Programming

# Project Report

For our project, the goal was to test the performance improvements that concurrent programming would yield in the context of digital processing. In particular, we were working with performing operations for enlarging and smoothing images using the Gaussian method in both a sequential and more concurrent manner and recording the time that it took for each implementation to complete their operations so that we could compare them. Digital image smoothing is commonly used to blend together pixels of an image in order to hide small imperfections and blemishes in the image. Specifically, the Gaussian blur that was implemented as the smoothing section of the program and is used as the pre-processing state for image enhancement algorithms by blending the pixels using the Gaussian function.

The program was first implemented to run in a sequential manner. This means that the program would take in the image file as input as perform smoothing and then enlargement in consequent order. At startup, the program runs through a central User Interface, which is implemented through JFrame. The user interface then allows the user to select the image to run the smoothing and enlargement on. Once this is done, the program will open another window with a re-sized version of the chosen file that is bound by the JFrame and allows the user to draw the desired rectangle over the image to specify the particular part of the resized image that he or she would like to perform a Gaussian blur on while the image is enlarged. In the case of the sequential method, most of the work for this algorithm is done within the files smoothing.java and Enlarge.java. At this point, the image that we are working with is split up into two different parts that we are working with. The smoothing method for the Gaussian blur

that we are using is implemented in the file GaussianBlur.java and is called by the smoothing.java file prior to the enlargement method calls in the file Enlarge.java. The specified area of the image marked by the user with the rectangle is stored as a variable called “captureRect,” and this is the area that the Gaussian Blur algorithm must perform work on. In the sequential implementation for our image processing algorithm, the program takes the entire selected region and loops through the entire selected region using a single thread to perform the desired Gaussian function on each of the pixels of the image. Using only a single thread, the algorithm would convert the “smoothingPartSource” to “smoothingPartTarget” by taking on the entire burden of performing a Gaussian Blur on all of the pixels of the captured rectangle. Once this is done, the program will take the smoothed portions on the image and re-insert it into “smoothingTarget” so that the result at this point in time is the same as the original image, but smoothing has been performed at the specified captureRect. At this point in time, the smoothing is complete and the algorithm sequentially continues to work by performing the enlargement method. The Enlarge.java file initializes the inputImage and outputImage and also initializes the enlargement ratio to 2, which we have kept in order to keep consistency of results. Similar to the Gaussian Blur algorithm, the Enlargement algorithm also loops through the entire image on a single thread. This algorithm loops through the entire image after it has been smoothed and re-creates it by increasing the ratio and coloring each pixel appropriately to create the final product of a new image that looks the same as the original input image, but has been smoothed at the appropriate area and has been enlarged by some specified enlargement ratio.

The following images sample what happens when the program is run. When the user selects a picture to perform the algorithm on, he or she is able to freely choose the captureRect as the position of the start point of the captureRect within the JFrame is displayed. Image 1 shows a sample of what the JFrame will look like when the user selects the captureRect in his or her image. Once processed, the program will return a new image that has been smoothed in the specified captureRect area and has

been enlarged by some enlargement ratio. This can be seen in Image 2, as the new image clearly shows a blur in the specified are of Image 1 and has also been enlarged.

**Image 1** **Image 2**



Simply by investigating the code and the way that it runs, it is plain to see that implementation of concurrency could improve the performance and runtime of the program. However, we saw that there could be many difficulties in trying to run the enlargement and smoothing methods at the same time without having to deal with several collisions. So we instead decided to take the approach of splitting up the image into parts when performing smoothing and enlarging and assigning multiple threads to each part to improve concurrency. As could be seen in the sequential implementation of the algorithm, each method required the program to run the specified method on each and every pixel of some portion of the image by itself. It is easy to see that this could be exhausting and time consuming on the program, particularly as the input sizes of the program increase with larger images and more

inputs. Therefore, we reasoned that the best way to start implementing concurrency and increasing the performance of the program would be to split up the image into multiple parts so that we can assign threads to perform the necessary work on each portion of the image. Since the input images are always of some quadrilateral shape and the captureRect that the users may draw is always a quadrilateral shape, it is plain to see that we are able to split our image into eight parts. Each of these parts may be assigned a thread to perform work, so we also assign a thread to each of these parts and thus split up the work that the program is doing and improve overall performance.

At this point, the program was changed to implement concurrency by including parallelism in the Gaussian blur and enlargement methods. We first decided to implement parallelism just in the Gaussian Blur method to see how much improvement we would see in the program. In the original file for the Gaussian blur, most of the work in the program was being done in two for() loops that go through each pixel in the captureRect and perform Gaussian function on them. In order to improve performance, the image must be split up into parts so that multiple threads can be assigned to each part of the image so that the program will not be overburdened as before. We created a new file "GaussianBlurParallel.java" to implement the desirable concurrency and initiated eight threads so that they can be assigned work. We begin by starting the first four threads and then joining them together and doing the same to the next four threads. The captureRect section of the image is then split up into four parts that we define by nested loops though the specified width and height of the image. For example, the first thread will perform operations on the first section of the captureRect as it loops from the start to width/2 and from start to height/2 and will apply the Gaussian function to each pixel in this region. In other words, thread 1 will apply the Gaussian function to the top-left section of the captureRect. The other threads will take care of other sections of the captureRect in a similar manner. This change was able to offer some increase in performance of the program, but we decided to take it further by applying the same kind of logic to the enlargement portion of the code. Similar to the

Gaussian Blur, the enlargement method suffers from some slowdown due to the fact that the program is assigned the entire area of the picture at once and must loop through the entire area to apply the enlargement algorithm to each pixel. In order to improve the performance of the enlargement method, we modified the "smoothing.java" file so that when the enlarging part of the program starts, we could assign subThreads to do work on the smoothing part of the program. Similar to what was done with the Gaussian blur method, we take out the smoothing part of the image and split up the subImage_out to four sections using their width and height properties and assign one subThread to each section of the image to perform the enlargement method. The same thing is done with the image that is to be returned when it is repainted at the end of the program.

Here, we include a table detailing the results of the runtimes of the sequential and concurrent programs. In order to keep the data consistent, each trial was run with a picture with size 2.9mb, a captureRect of size 1300*750, and an enlarging ratio of 2 run on the same machine with 2.6 GHz Intel Core i5, 8 GB Memory.

**Table 1**

| Trial | Sequential Time (ms) | Gaussian Blur Parallel (ms) | Gaussian Blur Parallel and Enlargement Parallel (ms) |
|---|---|---|---|
| 1 | 5538 | 5545 | 5513 |
| 2 | 5510 | 5337 | 4615 |
| 3 | 5505 | 5268 | 3192 |
| 4 | 5492 | 5108 | 3019 |
| 5 | 5486 | 5080 | 3001 |
| 6 | 5447 | 4971 | 2916 |

| | | | |
|---|---|---|---|
| 7 | 5435 | 4965 | 2746 |
| 8 | 5416 | 4943 | 2675 |
| 9 | 5404 | 4943 | 2673 |
| 10 | 5402 | 4942 | 2647 |
| 11 | 5381 | 4940 | 2640 |
| 12 | 5341 | 4911 | 2599 |
| 13 | 5265 | 4870 | 2583 |
| 14 | 5257 | 4790 | 2493 |
| 15 | 5174 | 4736 | 2471 |
| Average Time: | 5410.846154 | 5005.230769 | 2907.615385 |

By observing the data, we find that between the sequential algorithm and the algorithm that made the Gaussian Blur parallel, there was a speedup of about 8.104% between the average run-times of 5410.846 ms and 5005.2308 ms. Although the change is rather small, it is clear to see that there was some improvement in the performance of the image processing algorithm by implementing concurrency in portions of the code. The performance of the program can be further improved by parallelizing both the Gaussian blur and the enlargement methods. By comparison, there was a speedup of 86.092% between the sequential algorithm and the concurrent algorithm that implements parallelism in the Gaussian blur and enlargement method.