Parallel Programming Project - Bulk Image Enhancer Project

1st Anna Patricia B. Desembrana
College of Computer Studies
De La Salle University
Manila, Philippines
anna_desembrana@dlsu.edu.ph

2nd Clarissa Mae S. Mandadero
College of Computer Studies
De La Salle University
Manila, Philippines
clarissa_mandadero@dlsu.edu.ph

***Abstract*—Parallel programming consists of several techniques that allow the simultaneous execution of instructions through the decomposition of a task into smaller processes. This paper presents a bulk image enhancer created in Jupyter Notebook that implements the techniques, such as threads and locks, in order for the program to process each image in parallel. Results show that the application of such techniques have significantly improved the performance of the program with the number of threads being indirectly proportional to the processing time of the entire set of images.**

***Index Terms*—image enhancement, parallel programming, threads, processes, locks**

I. Introduction

Parallel programming is defined as the process of decomposing a problem into smaller tasks which can be executed simultaneously using multiple computing resources [1]. Its aim is to develop correct and efficient parallel programs which assign tasks to processes or threads. The synchronization and coordination of threads are required for the correct execution of the tasks [2]. Thus, through the proper implementation of parallel programming, a program can be executed more efficiently with a shorter wall-clock time through the use of more resources.

The goal of this project is to create a bulk image enhancer through the use of a parallel-based program. The program is implemented using Python programming language through the Jupyter Notebook. The image enhancement process is achieved through the use of the Python Imaging Library (PIL). PIL already has a module that is capable of image enhancement in terms of enhancing the brightness, sharpness, and contrast of an image [3]. While the threading module was used to create a parallel-based program [4].

Fig. 1 shows the bulk image enhancement process of this project. The program must be capable of accepting as input a set of images. Moreover, the following input will also be asked from the user: folder location of images, folder location of enhanced images, enhancing time in minutes, brightness enhancement factor, sharpness enhancement factor, contrast enhancement factor, and the number of threads to use. Upon receiving all needed input, the program will check each image's file format. The accepted file formats are jpg, png, and gif. If the file format of the image is incorrect, the image will not undergo the enhancing process. On the other hand, if the file format of the image is correct, image enhancement will be performed. The program will continue with the image enhancement until all images have been processed. The expected output of the program are the enhanced images that will be stored in a folder, and a text file containing the folder location of the enhanced images and the number of images enhanced.

The next section of this paper will discuss the program implementation of the bulk image enhancer through parallel programming. The third section will discuss the experiments performed and the results of the experiment. Lastly, the conclusion will briefly discuss how parallel programming techniques were used to improve the performance of the program in enhancing bulk images.

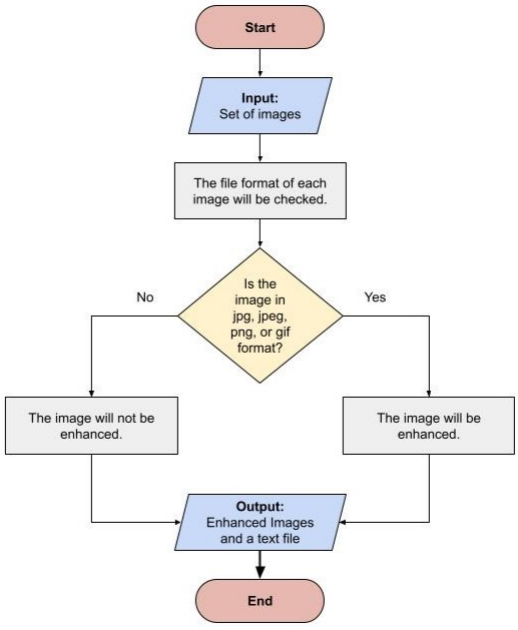

Fig. 1. Bulk Image Enhancement Process

II. Program Implementation

A. Image Enhancement

Image enhancement of the images were performed using ImageEnhance() module from PIL. First, all images are retrieved from the input folder, as seen in Fig. 2. Each image's file format is checked before it continues with the image enhancement process since the accepted file formats are only jpg, png, and gif. After retrieving all images, the image enhancement process was performed in the enhance() function, as seen in Fig. 3. The ImageEnhance() instances were initialized for each property: brightness, contrast, and sharpness. Afterwards, the image enhancement process is performed by stating the values for the three properties. The enhanced images are then saved to the output folder.

```
# Iterates through all images/files in
   the input folder
for i in os.listdir(path):
   ext = os.path.splitext(i)[1]
   #checks if the image is in jpeg, jpg,
      gif, or png format
   if ext.lower() not in valid_images:
      continue
   imageList.append((str(i),Image.open
      (os.path.join(path,i))))
```

Fig. 2. Code Snippet of Image Retrieval

```
# Initialize ImageEnhancers for each
   property
b = ImageEnhance.Brightness(currImg[1])
s = ImageEnhance.Sharpness(currImg[1])
c = ImageEnhance.Contrast(currImg[1])

# Enhance image
new_img = b.enhance(brightness)
new_img = s.enhance(sharpness)
new_img = c.enhance(contrast)

# Save image
new_img.save(output_folder + '/' + os.path
   .basename(filename) + file_extension)
```

Fig. 3. Code Snippet of Image Enhancement

B. Parallel Programming

Threads and locks from the threading module were used to create a parallel-based program. The creation and implementation of threads were performed in the createThreads() function, as seen in Fig. 4. A list of threads was created to allow parallel programming between multiple treads. After the creation of the list of threads, the start() function was called to allow all threads to perform the image enhancement process. Afterwards, the join() function was called which waits for all processes to complete before continuing with the execution of the rest of the code.

```
# List of threads implementation
for i in range(0, nThreads):
   t = threading.Thread(target=enhance,
      args=(imageList, outputFolder,
      brightness, sharpness, contrast,
      lock, startTime, timeLimit,
      shared_counter))
   threads.append(t)

# Start the execution of all threads
for t in threads:
   t.start();

for t in threads:
   t.join()
```

Fig. 4. Code Snippet of the Implementation of Threads

The implementation of locks was performed by creating a threading.Lock() object named lock in the createThreads() function which is passed on to the enhance() function. The lock object was used in the enhance() function to ensure that certain sections of the code are only accessible to one thread at a given time. The lock object was placed in two sections of the code: popping of the image, and incrementing the number of enhanced images, as seen in Fig. 5 and Fig. 6, respectively.

```
lock.acquire()
currImg = img.pop()
lock.release()
```

Fig. 5. Code Snippet of Image Removal Lock

```
lock.acquire()
IMG_CNT += 1
print("Saved " + filename)
lock.release()
```

Fig. 6. Code Snippet of Count Increment Lock

A lock was placed in the popping of the image, so that only one image is removed from the list at a specific time. While a lock was placed when incrementing the number of enhanced images to ensure that only one process gets to increment at a given time. This allows for a more accurate result because having multiple processes increment the number of enhanced images may result to its value becoming inconsistent.

III. Result

Two experiments were performed to better understand the performance of the program. In the first experiment, 500 images were enhanced. While in the second experiment, 5000 images were enhanced. In each experiment, the number of threads is manipulated to create 3 different test cases. The first test case used 1 thread, the second test case used 10 threads,

while the third test case used 20 threads. The results of each experiment with its 3 test cases can be seen in Table I and Table II.

TABLE I
EXPERIMENT #1: 500 IMAGES

No. of Threads	Trial #1	Trial #2	Trial #3	Trial #4	Trial #5	Average in secs
1	3.8761	4.2065	4.0519	4.0748	4.4011	4.1220
10	2.5898	2.1447	2.2062	2.2822	2.1488	2.2743
20	2.0786	2.1822	2.2149	2.3645	2.2738	2.2228

TABLE II
EXPERIMENT #2: 5000 IMAGES

No. of Threads	Trial #1	Trial #2	Trial #3	Trial #4	Trial #5	Average in secs
1	48.5830	44.3755	60.0067	45.1707	39.1331	47.4538
10	22.4451	21.6060	22.1852	22.8646	22.6705	22.3542
20	21.8664	22.4393	22.3456	20.5299	20.0318	21.4426

It can be observed that as the number of threads increase, the average processing time of the program decreases. This shows that increasing the number of threads means increasing the efficiency of the program. In Table I, the program that used 20 threads took the least time to process. The same result can be observed in Table II wherein the program that used the most threads (20) took the least amount of processing time.

In comparing the result of Table I and Table II, it can be seen that a huge difference in average processing time can be better observed in the experiment that used more images, 5000 images. There is a large 26-second difference between the program that only used 1 thread compared with the program that used 20 threads in experiment #2. Thus, it can be concluded that the larger the dataset to be used in the experiment, the more time can be saved.

One significant observation in the experiments is that there is very little difference in the average processing time of the program which used 10 threads compared with the program that used 20 threads. Thus, it can be concluded that for datasets that contain less than 5000 images, 10 is the most efficient number of threads when creating a parallel-based program as this will lessen the number of threads and resources to be used without greatly affecting the average processing time.

IV. CONCLUSION

Threads and locks were used in this project in order to separate the flow of execution of the image enhancement tasks. Instead of enhancing and saving each image one by one, the image enhancement is done in parallel through the use of threads. Moreover, the locks allow the synchronization of the processes in order to allow the threads to own the shared variable at a time, preventing interference. Results show that the implementation of the parallel programming techniques has significantly lessened the processing time of the bulk image enhancer. Moreover, the manipulation of the number of threads to be used affects the processing time. The more threads used in the execution of the processes, the shorter the processing time becomes.

REFERENCES

[1] B. Burns, "What is parallel programming?" Jan 2021. [Online]. Available: https://totalview.io/blog/what-is-parallel-programming

[2] G. Rauber, T. Rünger, *Parallel Programming*, 1st ed. Berlin: Springer Berlin, Heidelberg, 2013.

[3] "Imageenhance module." [Online]. Available: https://pillow.readthedocs.io/en/stable/reference/ImageEnhance.html

[4] "Threading - thread-based parallelism." [Online]. Available: https://docs.python.org/3/library/threading.html