International Journal of Multidisciplinary Research and Growth Evaluation.

Image compression in system through parallel & Sequential Execution

Babar Hussain [1], Tauqeer Ahmed [2]
[1-3] Department of Engineering and Computer Science, University of Messina, Piazza Pugliatti, 1, 98122 Messina ME, Italy

* Corresponding Author: **Babar Hussain**

Article Info

ISSN (online): 2582-7138
Volume: 03
Issue: 04
July-August 2022
Received: 09-06-2022
Accepted: 24-06-2022
Page No: 165-170

Abstract
The Parallel processing has become a significant tool for implementing high speed computing. For implementing this in image processing, several research and contributions have been done till now using several tools likes GPU (Graphical Processing Unit), CUDA (Computed Unified Device Architecture). The multicore computing is pervasive throughout most industries, and the image and machine vision industries are no exception. This could improve throughput and reduce response times for camera systems dealing with growing amounts of data. Images are processed using two or more computer cores in multicore image processing. In other words, the processing of a task from an imaging system is shared among numerous cores. Moving to a multicore system has the overall benefit of reducing response time and increasing throughput in an imaging system. Multicore allows users to make use of the latest PC processor designs, allowing algorithms and software to run quicker and perform more tasks.

Keywords: High performance computing, image processing, parallel computing, Python Programming, Image compression

Introduction

Image compression is a sort of data compression that is used to lower the cost of storing or transmitting digital photos. When compared to generic data compression approaches used for other digital data, algorithms can take advantage of visual perception and the statistical features of picture data to deliver greater outcomes. Image compression is a key stage in the field of image processing before we begin processing larger images or films. An encoder performs image compression and outputs a compressed version of the image. The mathematical transforms play a crucial part in compression operations. Particularly for the understanding, the parallelization is especially important as there has been significant growth in the amount of electronic multimedia shared by individuals over the internet, fueled improved internet access speeds and image/multimedia sharing websites such as social networking sites.

The parallel computing is having very important significance in several image processing techniques like edge detection, histogram equalization, noise removal, image registration, image segmentation, feature extraction, different optimization techniques and many more. The Parallel processing ‘employed to give simultaneous data-processing operations is used to represent a large class. In addition, a parallel processing system is capable of concurrent data processing to achieve faster execution times.

The increasing computational capacity and programmability of multi-core architectures offer promising opportunities for parallelizing image compression and processing methods.

Image is the two-dimensional distributions of tiny image points called as pixels. It can be considered as a function of two real variables, such as f (x, y) with f as the amplitude like brightness of the image at position (x, y). Image Processing is the process of enhancing and manipulation with an image to extraction of meaningful information. The Parallel computing or processing is the process of simultaneous uses of various compute resources to solve a computational job/task/work. The Main principle of parallel computing is to divide a task in such a way that the task executes in minimum time with maximum efficiency. To implement parallel computing there can be several kinds of parallel machine like a cluster of computers which is having multiple

PCs combined with an elevated speed network, a shared memory multiprocessor by connecting multiple processors to a single memory system.

Python Programming for Image Processing

Python interpreters are available for many operating systems that allow the execution of Python code on a wide variety of systems. an image is a rectangular grid of pixels with definite width and height. Each pixel has its own value. So, quality of image depends on this pixel values and pixel is the unit of information present in an image. Image Processing is the enhancement of images using mathematical operations for which the input is an image, such as a photograph or video frame and the output of image processing may be either an image or set of characteristics or parameters related to the image.

Python has multiple libraries for multiple purposes like web development, scientific and numeric computing, image processing. To work on images, Python has a library i.e., Python Imaging Library (PIL) for image processing operations. The Python Imaging Library provides many functions for image processing, the Python Imaging Library, or PIL for short, is one of the core libraries for image manipulation in Python programming language and is freely available on internet to download. Many of the image processing tasks can be carried out using the PIL library such as image inversion, binary conversion, cropping, writing text on images, changing intensity, brightness, image filtering, such as blurring, contouring, smoothing and many more. Its initial release was in the year 1995. And many versions of PIL are available according to our operating system. Some of the file formats that it supports are ppm, png, jpeg, tiff, bmp, gif. PIL has been written in C and Python programming language. PIL can be used for the image enhancement and the development of the Python based image processing application so that it becomes easy for the beginners to learn and understand the complex tasks of the image processing using Python based image processing. Python Image Library (PIL), using which we can prominently develop the Python based image processing software and can be useful for number of applications like remote sensing, agriculture, space center, satellites, medical and health sciences, etc. Thus, it can be concluded that Python and Image processing proves to be the better combination for learning, developing, and understanding the capabilities provided in it.

Decomposition Process

In this project our goal is to compare the parallel with sequential image compression, so the central thing we have done in this project decomposition means taking the data from one field, breaks up the data into smaller pieces, and assigns the pieces to different columns in the staging table. If we will see the result of this project as a digital image processing, so each image sample is quantized to a fixed number of bits and then the image is stored digitally as an array of bytes in a file representing the image pixels. Image compression technique is concerned the image array into another array of smaller size or representing the image file by another description of a smaller size. A decomposition technique is presented for compressing the color image. The compressed image is segmented into different regions based on global color properties. The segmented regions are decomposed and represented as unions of rectangles. The image which originally consist of a thousand of pixels, now consist of few hundred rectangles that can be stored using fewer bytes as compared to the original image.

a) Image Segmentation

An image segmentation is the process of allocating a specific label to every pixel available in an image so that the pixel with the same label can share the characteristics with each other for example the color or texture of an image. These segmentation covers the entire image. It can be represented as regions and objects.

b) Merging Technique

The artificial partitions generated by the quadtree decomposition need to be merged such that a group of connected rectangles should be represented as a union/region of the minimum number of possible rectangles. A region can be classified as s group of connected pixels exhibiting similar properties. The similarity between pixels can be in terms of intensity, color, etc. In region merging technique every pixel as an individual region. Normally, regions are selected as the seed region to check if adjacent regions are similar based of predefined rules. If they are similar, we merged them into a single region and move ahead to build the segmented regions of the whole image.

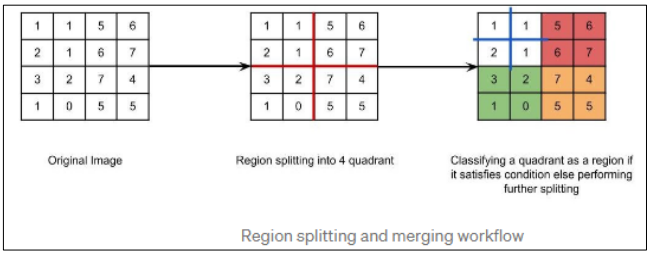

Fig 1: Merging techniques in Image Processing

$$|Z_{max} - Z_{min}| <= threshold$$

$$Z_{max} \rightarrow Maximum\ pixel\ intensity\ value\ in\ a\ region.$$

$$Z_{min} \rightarrow Minimum\ pixel\ intensity\ value\ in\ a\ region.$$

Fig 2: Segmentation algorithm for Image reconstruction

The algorithm used in the segmentation process has a parameter that can be used to control the accuracy of the image reconstruction.

Aggregation Process

In the project for image compression for parallel and sequential, the aggregation process takes data from multiple fields, aggregates the data, and assigns the aggregated data to a single column in the staging table, the aggregation of image processing is as it averages all the pixel values that contribute to the output pixel. In image processing a pixel aggregate method is working to define the output image values, based on a data sample, Aggregation process is to execute a sequence of steps in which pixels are gradually merged to produce larger and larger regions. In this section we focus on one step of such a procedure, in which a division of the image into a set of regions.

Mapping of Image process

Mapping is the new term of transformation used for allocating a point or region of one place to, another into an image. To compact the decorrelate the data, using differential coding, frequency/ sequency transforms, color transforms and principal components transforms. Mapping in image processing is an array of coordinates representing areas that you hyperlink to image's entire territory is hyperlinked to many destinations rather than simply the single destination within the image.

Quantization in Mapping for Image compression

The Quantization is involved in image processing, is a lossy compression technique achieved by compressing a range of values to a single quantum value. When the number of discrete symbols in each stream is reduced, the stream becomes more compressible. Quantization is required in mapping process often results in floating point data, can be uniform. In Digital image quantization is the process of determining which parts of an image can be discarded or consolidated with minimal subjective loss. Image quantization is inherently lossy however the image quality is reduced due to the loss of some information.

Mathematically Image Compression Methodology

In the field of Image processing, the compression of images is an important step before we start the processing of larger images or videos. The compression of images is carried out by an encoder and output a compressed form of an image. In the processes of compression, the mathematical transforms play a vital role. A flow chart of the process of the compression of the image can be represented as below flow chart.

Fig 3: Flow chart of the process of image compression

Methods of Image Compression

1) Applying the image transform (Mathematically)

It a function that maps from one domain (vector space) to another domain (other vector space). Assume, T is a transform, f(t):X->X' is a function then, T(f(t)) is called the transform of the function.

2) Quantization of the levels

It is a function of the location of the pixels. i.e., I (x, y) where (x, y) are the coordinates of the pixel in the image. So, we generally transform an image from the spatial domain to the frequency domain

3) Encoding the sequences

Generally, the newer level is determined by taking a fixed filter size of "m" and dividing each of the "m" terms of the filter and rounding it its closest integer and again multiplying with "m"

Description of Project Execution

Image processing is time-consuming. Real-time applications are frequently time constrained. As a result, serial image processing does not meet real-time requirements. Parallel computing approaches, particularly multicore and multiprocessing technologies, should be leveraged to overcome this problem.

A type of high-performance computing is parallel processing. Data is broken into small chunks and allocated to different execution units in parallel processing. As a result, data distribution is difficult in parallel processing. Data partitioning should be fine-grained to improve computer system efficiency by minimizing execution time and memory bottleneck.

The global interpreter lock (GIL) in Python causes single-CPU utilization by allowing only one thread to carry the Python interpreter at any given moment. The global interpreter lock was created to address a memory management problem, but as a result, Python can only use one CPU.

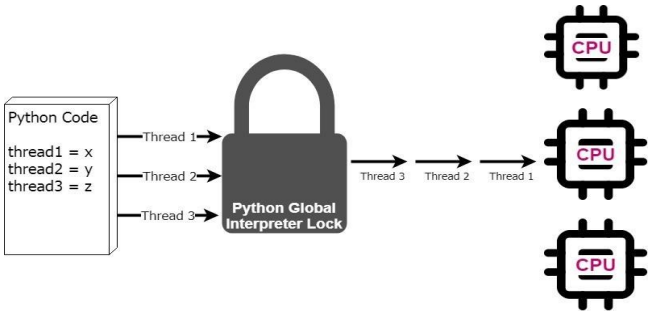

Fig 4: Representation of traditional serial python global interpreter lock

Multiprocessing can significantly speed up processing. Bypassing the global interpreter lock when running Python code allows us to take advantage of multiprocessing, which allows the function to run quicker. We can select some areas of code to circumvent the global interpreter lock and transmit the code to many processors for simultaneous execution using Python's built-in multiprocessing module. Multiprocessing has three prerequisites

a) The code must not be reliant on previous outcomes.
b) Data does not need to be executed in a particular order.
c) The program does not return anything that would need to be accessed later in the code.

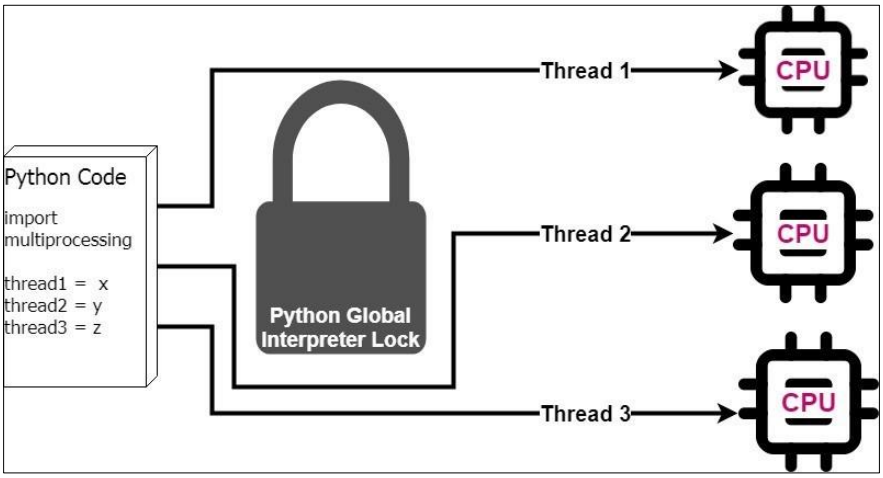

Fig 5: Representation of bypassing python global interpreter lock using multiprocessing

Classification and Method of Coding

The following section describes the code for image compression which has been performed serially and parallelly.

```
import os                    #perform operating system tasks, for creating and removing a directory (folder)
import time                  #function returns the number of seconds passed
import multiprocessing       #build parallel programs to implement multiprocessing-Process,Queue,Lock
from PIL import Image        #PIL is the Python Imaging Library which provides the python interpreter with image editing capabilities
import matplotlib.pyplot as plt      #visualizations
import numpy as np           #It provides a multidimensional array object, for various math operations.
imagesPath = os.path.join(os.getcwd(), "Images")
        #concatenated path components #returns current working directory of a process
```

```
 try: #block of code to be tested
 os.mkdir(os.path.join(os.getcwd(), "Serial Compressed")) #used to create a directory
 os.mkdir(os.path.join(os.getcwd(), "Parallel Compressed")) #used to create a directory
 except OSError as error:    #except block lets you handle the error
 pass #pass is a null statement
serialTime = {}     #serialization tool containing many serialization and deserialization shortcuts with timing
parallelTime = {}           #Parallel processing is a mode of operation where the task is executed simultaneously in multiple processors in the same computer
def compressImage(file, type, verbose = False):
#compression is minimizing the size in bytes of a graphics file
formats = ('.jpg', '.jpeg', '.png')
if os.path.splitext(file)[1].lower() in formats: #method in Python is used to split the path name into a pair root and ext
filepath = os.path.join(imagesPath, file)
 picture = Image.open(filepath)
compressedPath = os.path.join(os.getcwd(),type)
picture.save(os.path.join(compressedPath, file), "JPEG", optimize = True, quality = 10) #to save picture as jgp
return #run the rest of the main program
def serialProcess():                    #Serial processing is purely sequential
 print("==== Serial Image Compression ===\n")
 starttime = time.time()      #method of the time module returns the current system time
 for index,file in enumerate(os.listdir(imagesPath)): #loop over an iterable object and keep track of how many iterations have occurred
 print("\t"+str(index+1)+". Serial Compressing "+file)
 compressImage(file,"Serial Compressed") #call the function of Image compression
 if(index % 50 == 0 and index!= 0):
 serialTime[index] = time.time() - starttime
 global serialTimeTotal       #global keyword allows you to modify the variable outside of the current scope
 serialTimeTotal = time.time() - starttime
 print("\n\tSerial Image Compression Completed.")
def parallelProcess(): #Multiprocessing enables the computer to utilize multiple cores of a CPU to run tasks/processes in parallel
 print("\n=== Started Parallel Image Compression ===\n")
 starttime = time.time()
 global processes #global variable and make changes to the variable in a local context.
 processes = []
 for index,file in enumerate(os.listdir(imagesPath)): #loop over an iterable object and keep track of how many iterations have occurred
 print("\t"+str(index+1)+". Parallel Compressing "+file)
 if(index%50 == 0 and index!= 0):
 parallelTime[index] = time.time() - starttime
 p = multiprocessing.Process(target=compressImage, args=(file,"Parallel Compressed",))
 processes.append(p)          #adds a single item to the existing list p.start()
 global parallelTimeTotal #global keyword allows you to modify the variable outside of the current scope
 parallelTimeTotal = time.time() - starttime
 print("\n\tParallel Image Compression Completed.")
def plotGraph(): #to define the Plotgraph to visualize the serial and paralled process
 x = list(serialTime.keys())
 y1 = list(serialTime.values())
 y2 = list(parallelTime.values())
 X_axis = np.arange(len(x))
```

```
 plt.bar(X_axis - 0.2, y1, 0.4, label = 'Serial Processing')
 plt.bar(X_axis + 0.2, y2, 0.4, label = 'Parallel Processing')
 plt.xticks(X_axis, x)
 plt.xlabel("Images")
 plt.ylabel("Time Taken(s)")
 plt.title("Serial vs Parallel processing for Image
Compression")
 plt.legend()
 plt.show()
if __name__ == '__main__':
 serialProcess()
 parallelProcess()
 print("\n\n === Compression Statistics ===")
 print("\nNo. of images : 110")
 print("\nTotal size of images before compression : 276.5
MB")
 print("Total size of images after compression : 11.4 MB")
 print('\nSerial Compression took {}
seconds'.format(serialTimeTotal))
 print('Parallel Compression took {}
seconds'.format(parallelTimeTotal))
 plotGraph() print("Terminating all processes, This might
take a while ...")
```

```
 for process in processes: #Process. Queue. Lock
 process.join() #blocks the execution of the main process
until the process whose join method is called terminates
```

Function Descriptions

Compress Image (file, type): This function is used for compressing a given image using the python Pillow library. The function takes in 2 parameters, the first being the filename which is further used to fetch the file path. The second parameter is the type of processing being performed i.e., serial, or parallel, the type of parameter is used to save the resultant compressed file in the correct directory

Serial Process (): This function uses a simple for loop and serially calls the **compress Image ()** function to compress a given image from the selected directory. The function also captures the execution time for serial compression of the images. The traditional for-loop iteration goes through the list one by one and performs the function on each item individually.

Parallel Process (): This function uses the multiprocessing python library to send each task to a different processor. We use the multiprocessing module to create a new process for each list item and trigger each process in one call. We keep track of all processes by making a list and adding each process to it. After creating all the processes, the separate output of each core is combined and displayed together. The function also captures the execution time for parallel compression of the images.

Plot Graph (): This function makes use of the data stored in the 2 dictionaries initialized at the top. The dictionary consists of a key pair value where the key holds the number of images processed (n) and the value holds the time it took to run no. of processes. Using this data, the function plots a simple bar graph to compare the results of serial and parallel execution using the matplotlib python library.

Results of Image compression application

A sample set of 110 images were used in the program. Before compression, the combined size of the images amounted to 276.5 MB, after compression the combined image size drastically reduced to 11.4 MB. The above program gave the following output.

```
 === Compression Statistics ===

No. of images : 110

Total size of images before compression : 276.5 MB
Total size of images after compression : 11.4 MB

Serial Compression took 26.102900981903076 seconds
Parallel Compression took 9.633362531661987 seconds
```

Fig 6: Program Output

The multi processed code doesn't execute in the same order as serial execution. There's no guarantee that the first process to be created will be the first to start or complete. As a result, multi processed code usually executes in a different order each time it is run, even if each result is always the same. The following table shows the time taken for image compression run serially and parallelly for every 50 consecutive images.

Table 1: Time taken for Serial vs Parallel processing for n given images

No. Of Images	Time taken for Serial Processing (s)	Time taken for Parallel Processing (s)
50	5.31	0.66
100	14.03	1.18

As we can see, serial processing takes a considerably longer time to finish execution when compared to parallel processing. The total time taken to compress 110 images serially was 26.10 seconds, whereas the total time taken for parallel compression was just 9.63 seconds. Another observation made was that as the number of images increased, parallel processing performed significantly better.

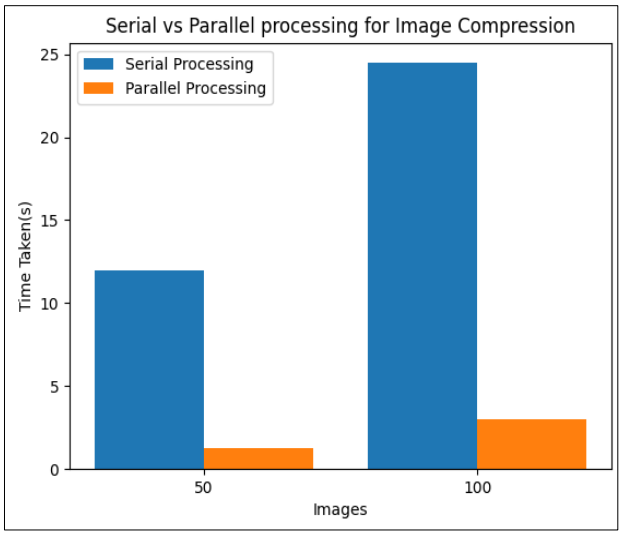

Fig 7: Serial vs Parallel processing for image compression

The few limitations observed were that the program didn't show any significant difference between serial and parallel processing for dual-core machines but showed a significant change for computers with a higher core count.

In the form of a comparative bar chart. Overall, for parallel execution, there was a time decrease of 23.91seconds. Thanks to multiprocessing, we cut down the runtime by 85.95% when

compared to the serial runtime. From this, we can conclude parallel processing significantly improves the run time of our program.

Conclusion

Parallel processing is rapider Image compression is significant because of the large image transfer happening on daily basis and it is better for the advancements in photography. Parallel is speedier Compression is useful because it helps reduce the consumption of expensive resources such as hard disk space or transmission bandwidth. The problem is to compress image files whilst maintaining the quality of the image. The advantage of our project is that compare the other image processing method with Parallel image processing, so we have good result that due to Parallel image compression, we have high frequencies are for the compression of images, this project uses techniques of parallelization to compress multiple images simultaneously with compared to sequential method.

References

1. Lee CK, Hamdi M. Parallel image processing applications on a network of workstations. Parallel Computing. 1995;21(1):137-160.
2. Rosenfeld A. Parallel image processing using cellular arrays. Computer. 1983;16(01):14-20.
3. Reeves AP. Parallel computer architectures for image processing. Computer Vision, Graphics, and Image Processing. 1984;25(1):68-88.
4. Chitradevi B, Srimathi P. An overview on image processing techniques. International Journal of Innovative Research in Computer and Communication Engineering. 2014;2(11):6466-6472.
5. López AF, Pelayo MC, Forero ÁR. Teaching image processing in engineering using python. IEEE Revista Iberoamericana de Tecnologias del Aprendizaje. 2016;11(3):129-136.
6. Rauber T, Rünger G. Parallel programming. Berlin, Germany: Springer; c2013.
7. Bogdanchikov A. Python to learn programming. Journal of Physics: Conference Series; c2013 .p. 423.