**Acknowledgement**

**Abstract**

We most likely have a large number of images on our PC. The trouble with having a large number of photos is that you'll wind up with duplicates. It's a good idea to keep track of your space. Duplicate picture detection from a collection of photographs is a time-consuming process that may be automated, and duplicate data can be deleted to save space. As we use our phones more, the number of undesirable duplicate photo and picture files grows at random in the device, ideally in each folder. Duplicate photographs and pictures eat up a lot of phone capacity and cause the phone to slow down. It's not possible to find and remove each image manually. We propose a unique strategy based on structural information degradation since human visual ability is insufficient to extract structure similarities from the naked eye. We construct a structural similarity index as a realistic answer to this problem and show it with a selection of photographs from our database. It can take a long time to find similar and duplicate photographs in these samples. In this case, duplicate photo finders come in handy. Finally, we'll evaluate the computation time and power required by many core threads vs. single core threads, as well as provide benchmarks and graphical representations for each.

**Table of Content**

**List of Figures**

**List of Tables**

**Abbreviations**

**Symbols and Notations**

1. **Introduction**

**1.1 Objective**

The main aim for us to make this project is that we want to make a platform to detect duplicate images from a large sample of given images in less time.

As our usage of mobile grows, the unnecessary duplicate photo and picture files grow in the device randomly, ideally every folder of the phone. The duplicate pictures/photos occupy lots of phone memory and also reduce the operating speed of the phone. Manually it is difficult to find and remove them.

Hence, we present the duplicate images detection using parallel processing from which you can scan through your entire phone and finds the duplicate image/picture/photo files for you

**1.2 Motivation**

No doubt we have numerous images on your computer and phones and even the cloud. It is difficult to keep a track of all images in the folder and the trouble increases because of various image sharing social media apps like Snapchat, Whatsapp, Instagram, Facebook. Thus we mix tons of images and cannot segregate it out. The trouble with having lots of pictures is that we automatically collect duplicates along the way and

At one point, due to memory constraints we need to delete duplicate files to keep the phone running in the desired state and store new images. This deletion process requires a lot of time. When we were searching about strategies which can reduce our time we found out that the duplicate images are scanned and compared one by one and thus the current methodology requires a high amount of time.

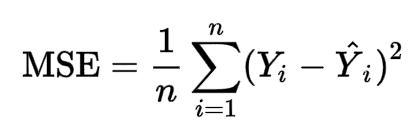
**1.3 Background Knowledge**

The technical knowledge involved in building a duplicate image detector using parallel computing is, python logics to find duplicates along with concepts of threading and divide and conquer.

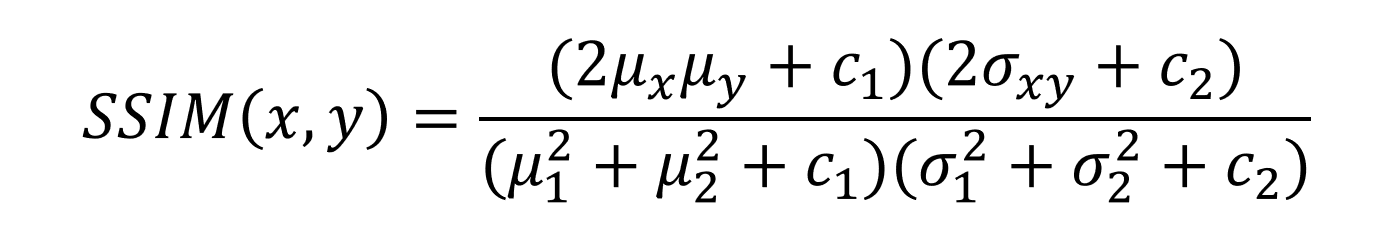
The basic image similarity concept involves use of MSE or Mean squared error, let us consider two images for which we want to compare them to determine if they are identical, or near-identical in some way.

We can perform this using a prominent technique as reviewed from the papers, it is to use algorithms such as Mean Squared Error (MSE) or the Structural Similarity Index (SSIM) which can easily be implemented through python libraries.

The Mean Squared Error approach takes all the image points or pixels and computes the error difference, this mean value helps in providing percentage similarity in the images. This index is not very useful in detecting similar images but very fast and accurate in identifying identical images since some similarities will always be observed in the case of similar background pictures.



To overcome the faults in the MSE approach SSIM approach was developed by renowned professor Wang et al. SSIM or Structural Similarity Index compares the images for similarity more than for identity which can obviously be extracted in the percentage similarity is a hundred per cent.



This method made it easy for us to determine if two images were the same or differed by slight image manipulation, compression, deliberate manipulation, or the addition of filters. We will extend the SSIM approach to include the differences between the images with OpenCV and Python. In particular, we draw bounding boxes around the regions in the two different input images. To calculate the difference between two images, we use the structural similarity index first published by Wang et al. in his 2004 article Image Quality Assessment: From Error Visibility to Structural Similarity. This functionality has been implemented in the scikit image library for image processing. The trick is to learn how to determine exactly where the image differences are in relation to (x, y) coordinate positions. The dependencies in the code involve an understanding of Python, OpenCV, scikitimage, and imutils.

**1.4 Computing Image Difference**

****

Fig 1.1. Manually inspecting the difference between two input images

If you notice the Fig 1.1 these two images, shape of the image are same the structures are same just the color scheme is different. The similarity between these images are very evident and can easily be identified by human eyes but, for a system to check such similarity in downloaded pictures is difficult. This demonstrates that sometimes image difference is very subtle and is noticeable by human eyes but sometimes better by a computer.

For example a missing component in either of the two images won’t be very visible to us immediately but computer will be able to find that similarity as well as difference. Such differences sometimes are important to compute in applications like phishing attacks which change a website logo with minute differences and trap users.

So the computer using SSIM and MSE approaches calculates difference between two images, image subtraction results in a new image matrix. This matrix value can be used to check mean squared difference to calculate percentage similarity. Also suppose we consider the same above example in Fig 1 then the image difference will give a blue output since all others will be present in the same ratio so on comparing the resultant matrix value program will be able to identify similarity in filters and image manipulations.

1. **Literature Survey**

**1.​ Composable Multi-Threading for Python Libraries**

**ADVANTAGE**​

As the composability layer for Python modules, Intel Threading Building Blocks is an open-source cross-platform toolkit for multi-core parallelism.

On multi-core computers, it aids in the unlocking of additional performance for numeric applications.

**DISADVANTAGE**

Because it limits the number of active threads, Intel TBB is ineffective for blocking I/O activities.

It's useful only for tasks that don't get stuck in the operating system.

**2.​ Near-Duplicate Detection of Images using Computer Vision Techniques**

**ADVANTAGE​**

The search is narrowed by clustering web image search engines and refining picture search results based on the search query. Organizing the results into distinct clusters makes surfing easier for consumers and helps them understand the NDD of images.

**DISADVANTAGE**

The NDD's runtime performance is a significant challenge. If the runtime is longer, a significant amount of computational resources is wasted. Traditional techniques' detection efficiency is inadequate since the NDD's uniqueness is enormous, and some "hot spot" photos in computer have too many copies while other photos have relatively few.

**​**

**3.​ Large-Scale Duplicate Detection for Web Image Search**

**ADVANTAGE​**

Because the existence of near-duplicates has a significant impact on search engine performance, vision-based algorithms and feature extraction methods for the detection of near duplicate images have been developed.

**DISADVANTAGE**

The NDD's runtime performance is a significant challenge. If the runtime is longer, a significant amount of computational resources is wasted.

**4.​ Node based multithreaded open source robotics**

**ADVANTAGE​**

Provides a standards-based and modular platform. A publish-subscribe object request broker lies at the heart of the system architecture. This allows common Unix knowledge and experience to be reused.

**DISADVANTAGE**

Complex programming.

**5.​ Parallel Divide and Conquer**

**ADVANTAGE​**

Algorithms for the parallel programming paradigm that are generic. It's possible to adapt it to divide and conquer. As a tree machine divides a problem repeatedly, the wave of computing travels sequentially across the levels of the tree, increasing the degree of parallelism exponentially.

**DISADVANTAGE**

Every node in a tree machine with p processors holds an array of size n when it solves a problem of size n. As a result, the size of the problem is constrained by the memory of a single node.

To save sorting time, a tree computer divides a sorting problem into smaller segments and employs only half of its processors (the leaves).

**6. Wang, Z., Bovik, A. C., Sheikh, H. R., & Simoncelli, E. P. (2004). Image quality**

**assessment: from error visibility to structural similarity. IEEE transactions on**

**image processing, 13(4), 600-612.**

**ADVANTAGE**

Recommended using structural similarity as a motivating principle for the development of picture quality measurements. To showcase our structural similarity notion, we created the SSIM index and demonstrated that it outperforms other approaches in accounting for subjective quality measurements of 344 JPEG and JPEG2000 compressed images in our experiments.

**DISADVANTAGE**

First, several image processing algorithms must be investigated in order to optimise the SSIM index.

Second, the SSIM index's application scope may not be limited to image processing and may be used to compare any two signals as a similarity measure.

**7. Hosoi, T., Kobayashi, K., Ito, K., & Aoki, T. (2011, September). Fast image**

**inpainting using similarity of subspace method. In 2011 18th IEEE International**

**Conference on Image Processing (pp. 3441-3444). IEEE**

**ADVANTAGE**

In the learning step, the proposed method, image impainting, generates the subspace from many images related to the object class and estimates the missing pixel values of the input image that belong to the same object class in the inpainting step to maximise the similarity between the input image and the subspace.

**DISADVANTAGE**

Borders of coherent impainted regions are subject to blurring. Taking entire blocks of audio/video information is not recommended.

**8. Singh, N., Browne, L. M., & Butler, R. (2013). Parallel astronomical data processing with Python: Recipes for multicore machines. Astronomy and Computing, 2, 1-10.**

**ADVANTAGE**

PyCOMPSs, a framework for developing parallel computational workflows in Python, is presented in this paper. In this method, the user writes her script sequentially and then decorates the functions so that they can be launched as asynchronous parallel processes. The script's intrinsic concurrency is used by a runtime system, which detects data dependencies between jobs and spawns them to the available resources.

**DISADVANTAGE**

Python is an interpreted language and dynamically-typed language hence slow in speed.

**9. Saha, A. (2012). Parallel programming in C and Python. Linux journal, 2012(217),**

**ADVANTAGE**

High-level parallel programming with Python and the BSP model is given special emphasis. The fundamentals of BSP are discussed, as well as how it differs from other parallel programming tools such as MPI. Then, using high-level library design and mixed-language (Python-C or Python-Fortran) programming, they show how to use Python and BSP to solve a partial differential equation from computational science.

**DISADVANTAGE**

Only a small portion of the code is time-critical, necessitating the use of a compiled language.

**10. Narayanan, S., & Thirivikraman, P. K. (2015). Image similarity using fourier transform. Journal Impact Factor, 6(2), 29-37.**

**ADVANTAGE**

For picture registration, a similarity measure based on values from their respective Fourier Transforms is proposed in this study. The method generates signatures based on image content rather than image annotation, and hence does not require human intervention. It computes the final rank for measuring similarity using both the real and complex components of the FFT.

**DISADVANTAGE**

Any reliable method must precisely represent all objects in a picture, and different strategies may be required depending on the size of the image data collection.

1. **Technical Specification**

**3.1 Software Specification**

Visual Studio Code

Github

Python

**3.2 Hardware Specification**

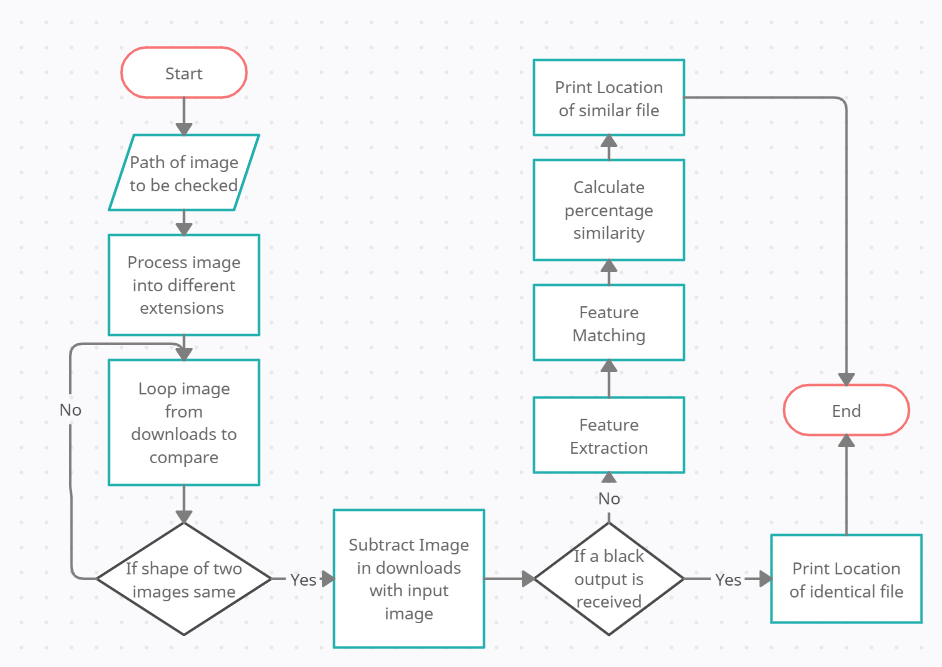
4 cores in the PC

12 GB RAM

I5 Intel Processor

500GB Storage

1. **Design**

****

1. **Proposed System**

**5.1 Existing and Proposed Solutions**

**The trouble with having lots of pictures is that we end up collecting duplicates. There are a lot of duplicate image finders listed to get rid of duplicate and similar photos. It would be a wise thing to manage space efficiently.**

**Existing Solution:**

**1. Duplicate Photo Finder**

**2. Anti-Twin**

**3. VisiPics**

**4. Similar Image Search**

**5. Awesome Duplicate Photo Finder**

**Proposed Solution:**

**1. An open source Py package to remove duplicate Images**

**5.2 Platforms and tools used**

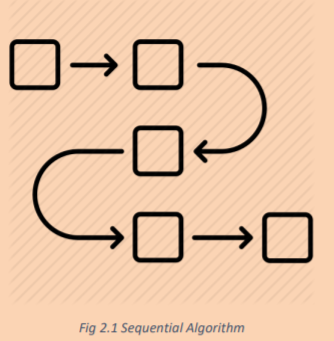
* PyCharm
* Visual Studio code
* Github
* Python3
* Python modules and libraries(OpenCv, multiprocessing, threading)

**5.3 Program Description**

We have created a working Script to detect duplicate images which identifies duplicate images recursively using multiprocessing library of python.The Structural Similarity Index which was first introduced by Wang et al. in their 2004 publication Image Quality Assessment: From Error Visibility to Structural Similarity, will be used to quantify the difference between two images. This approach is already included in the scikit-image image processing library. The trick is to figure out how to pinpoint exactly where the image differences are in terms of (x, y)-coordinate position. To do so, we'll need to ensure that our system has Python, OpenCV, scikit-image, and imutils installed.

**5.4 Serial Algorithm**

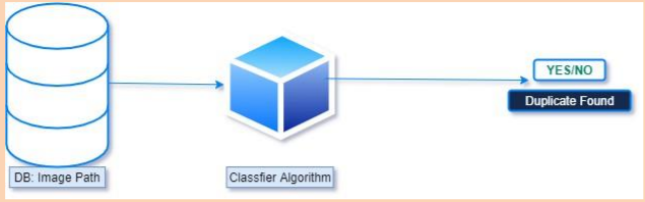
Search for Images Recursively from a given starting directory Compare two images by their RGB values and find image similarity using cv2 library sequentially.

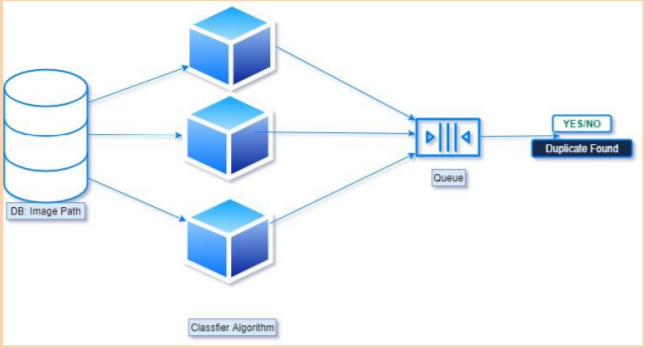


A sequential algorithm, also known as a serial algorithm, is a computer programme that is run sequentially, from start to end, without any additional processing, rather than concurrently or in parallel. The phrase is mostly used to contrast with concurrent or parallel algorithms; most common computer algorithms are sequential algorithms, even if they aren't explicitly labelled as such because sequentialness is a given. Concurrency and parallelism are two different notions in general, although they frequently overlap – many distributed algorithms are both concurrent and parallel. The sequential algorithm contrasts the two without specifying which is which. If you need to distinguish between them, utilise the opposing pairings sequential/concurrent and serial/parallel.

**5.5 Parallel Algorithm**

Here we use the divide and conquer method.





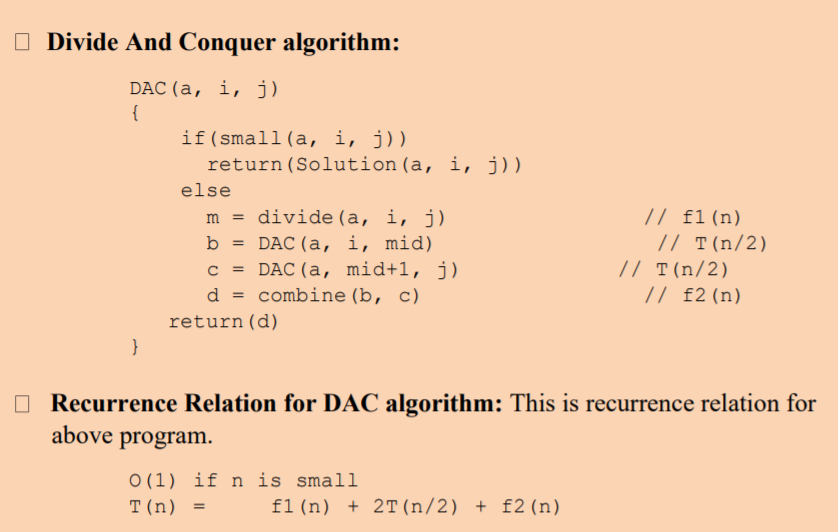
**5.6 Divide and Conquer**

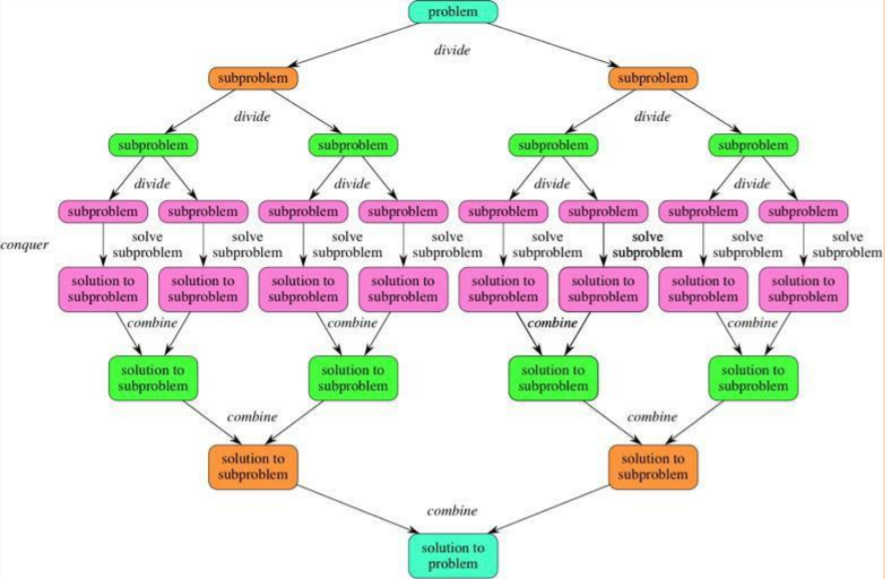
This technique has three parts:

1. Divide: dividing the problem into sub problems.

2. Conquer: calling sub problem recursively until sub problem solved.

3. Combine: The Sub problem Solved so that we will get find problem solution.





**5.7 Parallelisation using multiple processors**

Search for Images Recursively from a given starting directory Compare two images by their RGB values and find image similarity using cv2 library by spawning processes using multiprocessing library API

Contexts and start methods

Multiprocessing supports three ways to start a process. These start methods are:

**1.Spawn**

A new python interpreter process is started by the parent process. Only the resources required to operate the process object's run() method will be passed down to the child process. Unnecessary file descriptors and handles from the parent process, in particular, will not be inherited. When compared to fork or forkserver, this technique takes a long time to start a process.

**2. Fork**

The Python interpreter is forked by the parent process using os.fork(). When the child process starts, it is nearly identical to the parent process. The child process inherits all of the parent's resources. Note that forking a multithreaded process safely is difficult.

**3. Forkserver**

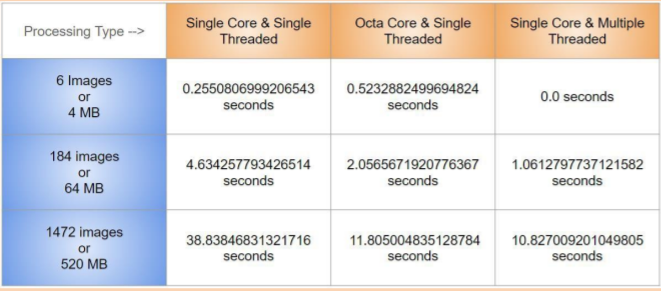
In the forkserver start method, a server process is started. When a new process is required in the future, the parent process connects to the server and requests that it fork a new one. Because the fork server process is single threaded, using os.fork is safe (). There are no resources left over that aren't needed.

**5.8 Parallelisation using threading**

Python does include a Threading library. Many programmes, especially those involving network programming or data input/output (I/O), are network- or I/O-bound. This indicates that the Python interpreter is waiting for the outcome of a function call that manipulates data from a "remote" source, such as a network address or a hard disc. Reading from local memory or a CPU cache is far faster than this. As a result, if several data sources are being visited, one way to speed up such code is to create a thread for each data item that needs to be retrieved. Consider a Python script that scrapes a large number of web URLs. Given that each URL will have a download time that exceeds the computer's CPU processing power, a single-threaded approach will be heavily I/O bound.The code may download many data sources in parallel and merge the findings at the conclusion of each download by creating a new thread for each download resource. This implies that subsequent downloads are not held up by previous web page downloads. Instead of being bound by the bandwidth constraints of the client/server(s), the application is now bound by the bandwidth restrictions of the client/server(s). Many financial applications, on the other hand, are CPU-bound due to their large numerical demands. Monte Carlo simulations, for example, frequently employ large-scale numerical linear algebra solutions or random statistical draws. As a result, there is no benefit to using the Python Threading module for such jobs in terms of Python and the GIL. Python provides four options for dealing with this. To begin, you can use the multiprocessing module to run functions in parallel. Threads are a second alternative to processes. The Python threading module is a good place to start for more information. Third, you can use the system() method of the os module or methods provided by the subprocess module to call external applications and then collect the results. The multiprocessing module includes a great set of mechanisms for handling simultaneous routine execution. Processes, agent pools, queues, and pipes are all examples of this.

**6.Results and Discussion**

**6.1 Analysis: Execution Time vs Data Size**

****

**Note: Threads are not synchronized, after sync time taken for 520MB is 13.5 sec using 100 Threads**

**6.2 Measurement Methodology**

**OS -**  Windows 10

**Value Calculation :** Mean of 3 exception

**Data Used:-** Consistent with all Scripts

**DataSelection:-** No Preference

**Error Prone areas and why:-**

Image Processing TimeSheet: It’s possible that processor or thread worked on bigger images first

**6.3 Explanation for Behaviour: MultiProcessing in PY**

1. **Multiprocessing adds CPU to increase computing power.**
2. **Multiple processes are executed concurrently**
3. Creation of a process is time-consuming and resource intensive.
4. Multiprocessing can be symmetric or asymmetric.
5. The multiprocessing library in python uses seperate memory space, multiple CPU cores, bypasses GIL limitation in CPython,child processes are killable and is musch easier to use.

**6.4 Explanation for behaviour: MultiThreading in PY**

1. Multithreading creates multiple threads of a single process to increase computing power
2. Multiple threads of a single process are executed concurrently.
3. Creation of a thread is economical in both sense time and resource
4. The multithreading library is lightweight, shares memory, responsible for responsive UI and is used well for I/O bound applications.
5. The module isn’t killable and is subject to the GIL
6. Multiple threads live in the same process in the same space, each thread will do a specific task, have its own code, own stack memory, instruction pointer, and share heap memory.

**6.5 Explanation for behaviour: Explained with Data**

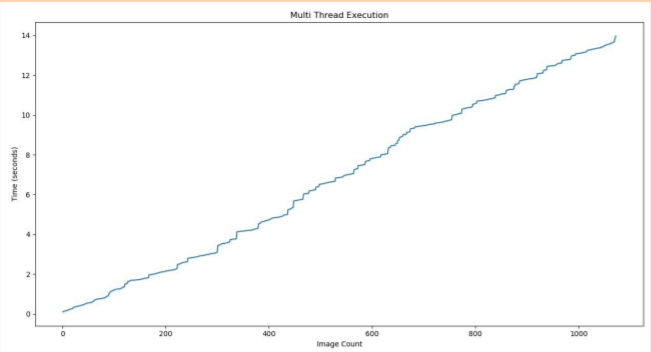
1. For Smaller Sample Thread work Better but on creation of too many threads then comes context switch in play in play which acts as a barrier
2. But Creating a Thread is a cheap process when creating limited number of threads as they all are sharing same resources and hence quite faster than Multiple Processinng at that time.
3. Multiple processor have their own resource bucket, they are GIL safe and can work in parallel because of this reason with increase in data they worked better than threading.

| Image Number | Single Core | Multiple Core | Multiple Threads(100) |
| --- | --- | --- | --- |
| 1 | 0.16798 | 0.10107 | 0.09719 |
| 2 | 0.32265 | 0.19383 | 0.12229 |
| 3 | 0.46663 | 0.31848 | 0.13236 |
| 4 | 0.58964 | 0.47482 | 0.13739 |
| 5 | 0.76592 | 0.65649 | 0.13739 |

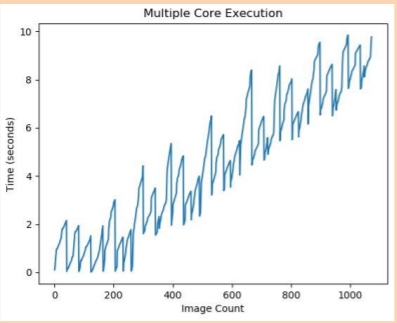
**6.6 Analysing Multiprocessing V/S MultiThreading V/S Single Core Execution.**

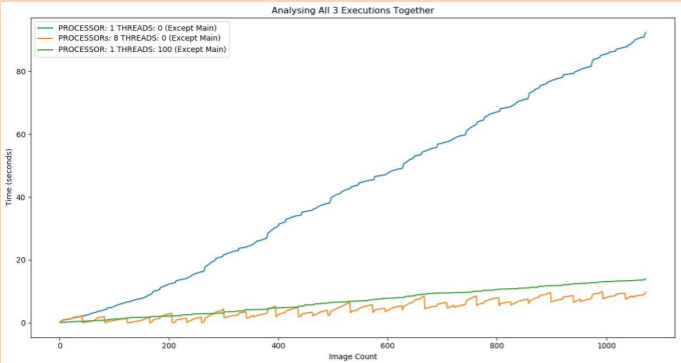
****

**Image VS Time Graph For Single Core Execution.**

****

**Image VS Time graph for Multi Thread Execution.**

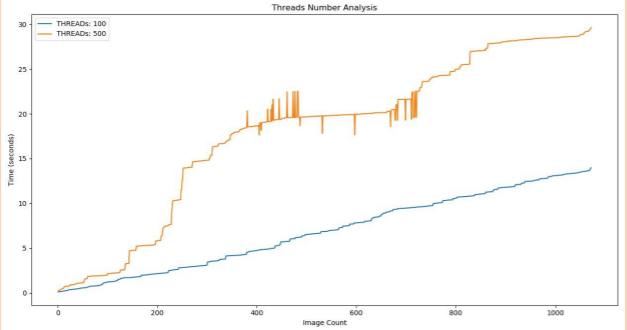
****

****

**Analysis of all 3 Executions.**

**Explanation.**

We can clearly observe here that the Time Taken for Single Core Execution increases exponentially with time. But for Multiprocessing and MultiThreading, it increases with a little slope value. An addition to the observation is the fact that the Multiprocessing Library is beating the thread as data is increased this is because Multiprocessing is resources extensive and so takes higher time to start its processing but then it processes the data like a charm whereas Thread have interrupts which is continuously degrading the performance.



How many threads are too many?

**Explanation**

When the no. of threads were increased the total time to process decreased but what we observe is that because Interrupts things get worsen and it instead takes more time to read the data and when threads were further increased to around 5000 then the output had around 586/1072 outputs generated and else went missing..

So by above observation it can be concluded that higher the threads add to the overhead of thread creation and the interrupts degrade the performance.

**Some Interesting Findings:-**

Sync Thread Script with 5k threads

1. Time Taken : 10.31 sec
2. Time Taken to process 1 image : 2.16 sec
3. Strange finding:- Only 586/1072 were processed.

**7.Conclusion**

Use of Threading is enough and more suited in Applications like GUI and Networking as it’s less resource intensive, Both Multi Processing and MultiThreading have similar performance in I/O bound operations.

Since Most online servers provide with single core only so Threaded apps are more used but for Research purposes we need more computation power and parallel execution with less complication and there comes Multiple Processors in to rescue.

References