# COMPARATIVE ANALYSIS ON SERIAL COMPUTING VERSUS PARALLEL COMPUTING

**Prepared For**

Operating Systems Reports

**Prepared by**

Hamna Siddiqui

**February 9, 2022**

## Index

## 1. Introduction

In this contemporary, growing and fast moving era; efficient, coherent and systematic computations are a necessity. Earlier, most of the computer programs and software were designed and standardized for serial computing with a view to execute processes/programs one by one, to solve a problem algorithm divides it into smaller discrete instructions to process them one after one further. The major drawback with the serial computing was these discrete instructions and its one after one execution which was taking an abundance of time and was affecting overall execution speed.

In order to tackle these issues and make executions efficient and optimized along with momentary time and money saving benefits via big data; serial computing is being transformed into parallel computing. The gist of our research is that parallel programming can reduce complexity of the programs via dividing problems into small instructions and solves them simultaneously as each applied resource is working in parallel.

The report has certain enlightenment on parallelization analysis and its outcomes along with the best possible implementation.

## 2. Body of report

The purpose of this report is to provide a detail hands-on analysis of the algorithm's efficient performance.

This report points out comparative analysis and distinguishes between time calculations of selected algorithm in serial computing and in parallel computing along with relevant implementation approaches and requirement analysis to tackle them effectively and compute optimized speed.

The aim of this report is to find answers for the following questions:

- How parallel computing is better than serial computing ?
- What is the time and speed difference between them ?

### 2.1 Specifications and requirements analysis

**1) For Image Processing Algorithm:**

This image processing algorithm degrades high resolution images using Python Imaging Library (PIL) by applying the GaussianBlur method on each image.
Inputs
A list containing relative paths of all the high resolution images
The time taken by image processing algorithm to completely execute can be reduced significantly by using parallel computing. We can create a separate process for the processing of each image by using the Process class from the python's multiprocessing library.

**2) For Image Downloading Algorithm:**

In image downloading algorithm we imported threading for parallelization purposes primarily because of the large data and I/O bound requirements. A request module has been imported to retrieve a data from a specified URL; It works as a request response protocol between a client and a server. We imported a time to return the total number of seconds passed in processing the whole code.

**3) For Banker's Algorithm:**

We imported random integers and floor from math to generate and return large random numbers array along with time to calculate total processing time and we did parallelization via threading because it's an I/O bound while the multiprocessing is CPU bound and if we have high competition algorithm (with deep mathematical calculations then we use multiprocessing) while when we have a large data to process then we use threading because in loading data time is being taken by the algorithm and that is mainly I/O bound.

## 2.2 Description and performance evaluation

### 1) For Image Processing Algorithm:

Steps of serial computing:
Firstly, the program loads images from the “\img” directory where all the high resolution images are located. Applies gaussian blur of variable amount on each image, and finally stores the processed image in the “\processed” directory. Both of the directories i.e “\img” and “\processed” are located relative to the running program script.

Problem in serial computing
In serial computing, as the quality of each image is high, the process takes a long time to apply the conversion on each image one by one. In result, the overall program executes using a single process while consuming a lot of time.

Steps of parallel computing
To process all the images separately in their respective processes, a for loop is used to take out each path of the image one by one from the images path list, passing it into the Process class, which has been imported from the python’s multiprocessing library, and storing each instance of the Process class in a processes list. The Process class targets the function which takes in argument the path of the image, processes the image, and saves it into the processed directory. Finally, each instance of process in the processes list is started and joined using two separate for loops.

**Performance Evaluation:** By implementing parallel processing in the image processing algorithm, we achieved parallel computing, hence the time taken by the program to fully execute is reduced remarkably. From 23.8 seconds to 11.1 seconds.

### 2) For Image Downloading Algorithm:

In an array we stored some urls to download pictures, then we stored a starting time in a variable and made a function with an argument of url’s array variable; we retrieved the data via provided url and opened the file with a function (with open) to write in binary mode and starts printing the file after downloading it completely.

From Threading loading of large data with efficient time is being used to manage and create threads, it is executing the task of downloading images via appending it in the end with an argument of url’s array; and combining it along with a resulting time speed.

**Performance Evaluation:** In this algorithm the images are being downloaded from internet on runtime and it is overall calculating the processing time for the downloading completion in parallel and in serial. So far we observed that the output we got in parallel processing by using threading is much more efficient than the serial processing.

The speed of parallel is: 12.02 seconds while the speed in serial is: 82.07 seconds

**<u>3) For Banker's Algorithm:</u>**

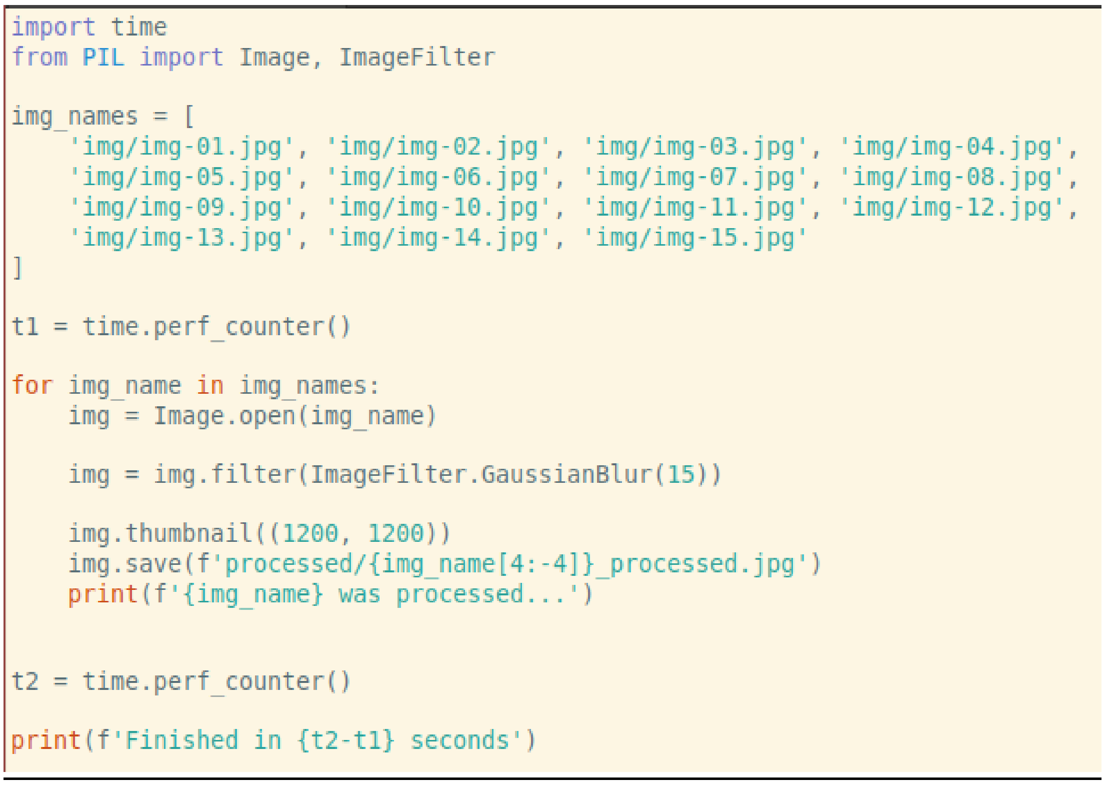
For no. of processes we used n and for no. of resources we used m as variables with a value of 5000 each. An Allocation and max matrix is being generated via randint with a lower and higher limit and dividing processes an resources into 2 halves in a for loop to iterate accordingly, then we divided this number into two (left and right to process in parallel) and used for loops for I and j and calculated need via subtracting allocation from max in both functions. After alloc and max matrixes generation and calculations of need matrix we joined both needs and a time is being calculated by subtracting start from finish.

**Performance Evaluation:** Its speed has been modified remarkably; in parallel its total time is 4.3 seconds which is visibly better than serial which was 4.99.w

## 3. Evidences

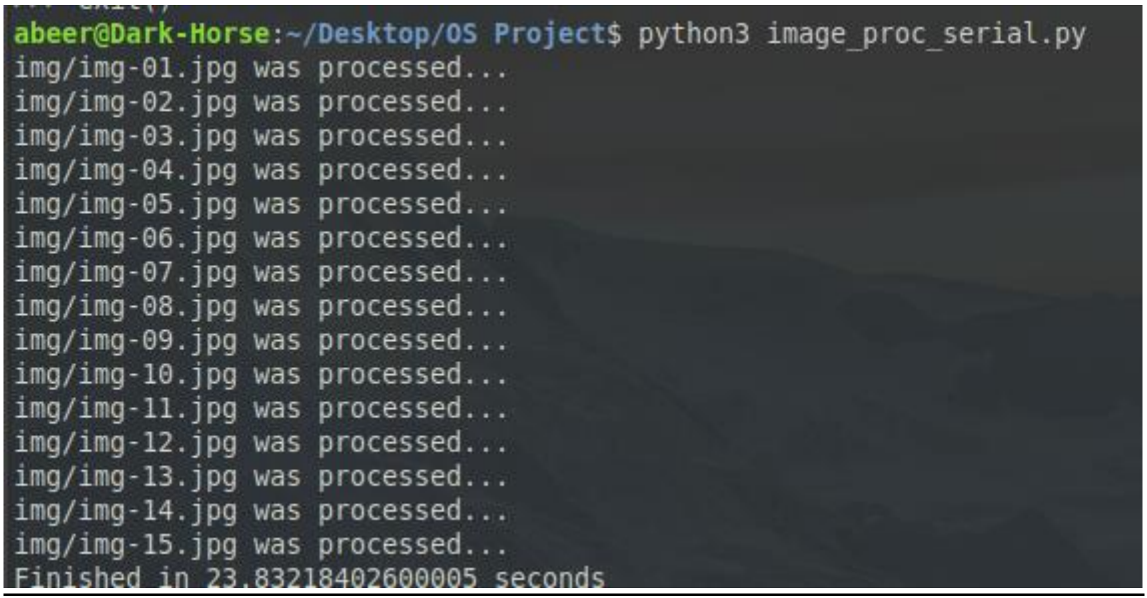
### 3.1 Screenshots of results and programs

<u>Image Processing Algorithm IN serial:</u>



```
import time
from PIL import Image, ImageFilter

img_names = [
    'img/img-01.jpg', 'img/img-02.jpg', 'img/img-03.jpg', 'img/img-04.jpg',
    'img/img-05.jpg', 'img/img-06.jpg', 'img/img-07.jpg', 'img/img-08.jpg',
    'img/img-09.jpg', 'img/img-10.jpg', 'img/img-11.jpg', 'img/img-12.jpg',
    'img/img-13.jpg', 'img/img-14.jpg', 'img/img-15.jpg'
]

t1 = time.perf_counter()

for img_name in img_names:
    img = Image.open(img_name)

    img = img.filter(ImageFilter.GaussianBlur(15))

    img.thumbnail((1200, 1200))
    img.save(f'processed/{img_name[4:-4]}_processed.jpg')
    print(f'{img_name} was processed...')


t2 = time.perf_counter()

print(f'Finished in {t2-t1} seconds')
```

Output:

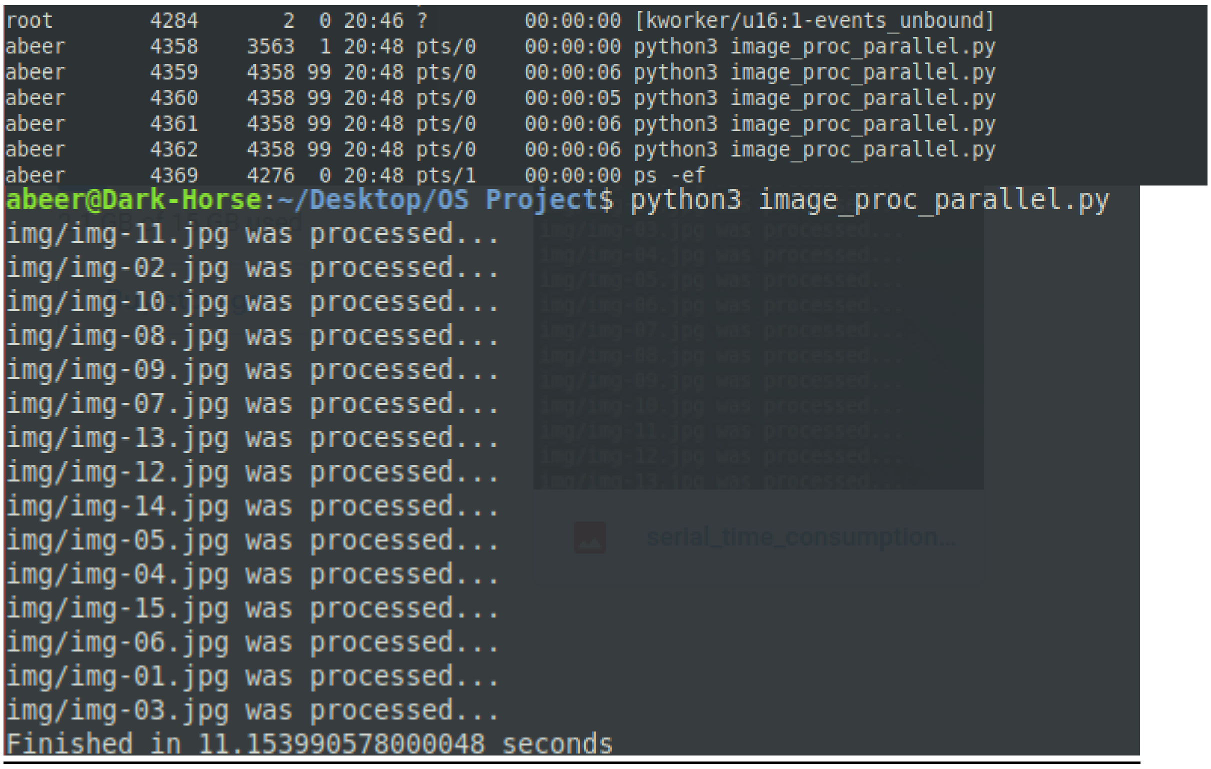


```
abeer@Dark-Horse:~/Desktop/OS Project$ python3 image_proc_serial.py
img/img-01.jpg was processed...
img/img-02.jpg was processed...
img/img-03.jpg was processed...
img/img-04.jpg was processed...
img/img-05.jpg was processed...
img/img-06.jpg was processed...
img/img-07.jpg was processed...
img/img-08.jpg was processed...
img/img-09.jpg was processed...
img/img-10.jpg was processed...
img/img-11.jpg was processed...
img/img-12.jpg was processed...
img/img-13.jpg was processed...
img/img-14.jpg was processed...
img/img-15.jpg was processed...
Finished in 23.83218402600005 seconds
```

Image Processing Algorithm IN Parallel:

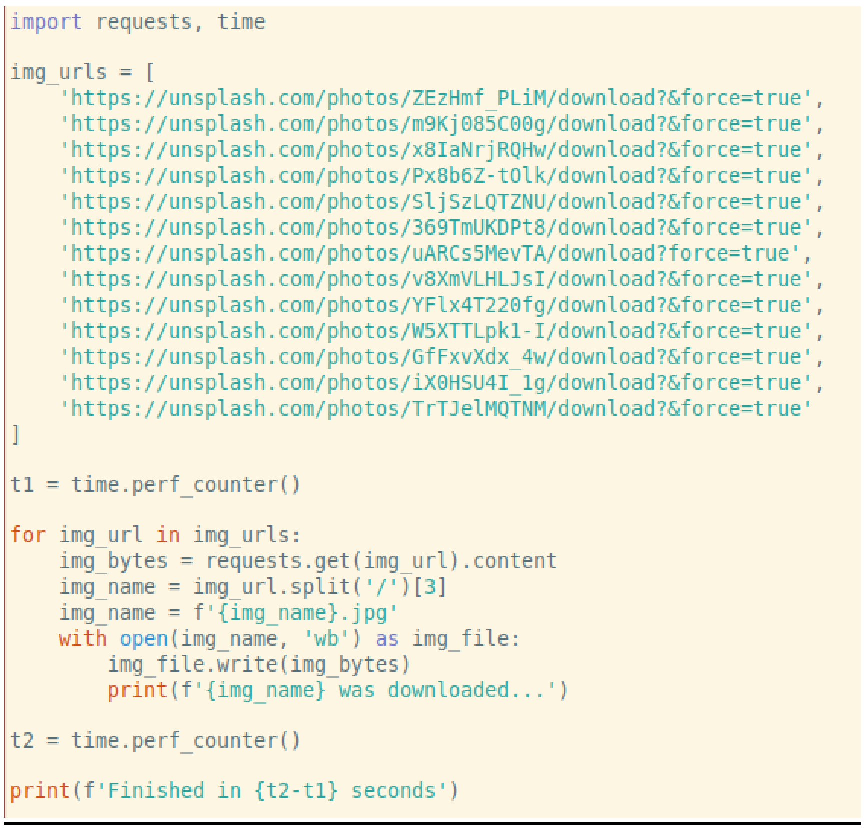
```python
import time
from multiprocessing import Process
from PIL import Image, ImageFilter

img_names = [
    'img/img-01.jpg', 'img/img-02.jpg', 'img/img-03.jpg', 'img/img-04.jpg',
    'img/img-05.jpg', 'img/img-06.jpg', 'img/img-07.jpg', 'img/img-08.jpg',
    'img/img-09.jpg', 'img/img-10.jpg', 'img/img-11.jpg', 'img/img-12.jpg',
    'img/img-13.jpg', 'img/img-14.jpg', 'img/img-15.jpg'
]

t1 = time.perf_counter()

def process_images(img_name):
    img = Image.open(img_name)

    img = img.filter(ImageFilter.GaussianBlur(15))

    img.thumbnail((1200, 1200))
    img.save(f'processed/{img_name[4:-4]}_processed.jpg')
    print(f'{img_name} was processed...')

processes = []

for img_name in img_names:
    processes.append(Process(target=process_images, args=(img_name, )))

for proc in processes:
    proc.start()

for proc in processes:
    proc.join()

t2 = time.perf_counter()

print(f'Finished in {t2-t1} seconds')
```

Output:



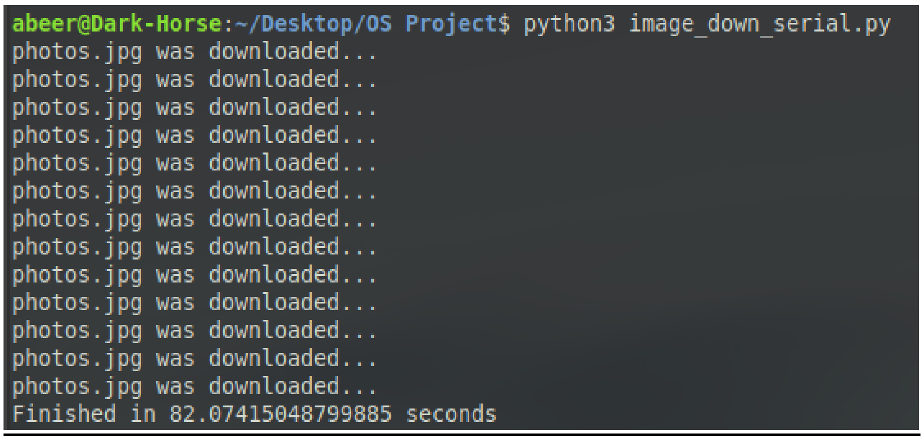
```
root         4284       2  0 20:46 ?        00:00:00 [kworker/u16:1-events_unbound]
abeer        4358    3563  1 20:48 pts/0    00:00:00 python3 image_proc_parallel.py
abeer        4359    4358 99 20:48 pts/0    00:00:06 python3 image_proc_parallel.py
abeer        4360    4358 99 20:48 pts/0    00:00:05 python3 image_proc_parallel.py
abeer        4361    4358 99 20:48 pts/0    00:00:06 python3 image_proc_parallel.py
abeer        4362    4358 99 20:48 pts/0    00:00:06 python3 image_proc_parallel.py
abeer        4369    4276  0 20:48 pts/1    00:00:00 ps -ef
abeer@Dark-Horse:~/Desktop/OS Project$ python3 image_proc_parallel.py
img/img-11.jpg was processed...
img/img-02.jpg was processed...
img/img-10.jpg was processed...
img/img-08.jpg was processed...
img/img-09.jpg was processed...
img/img-07.jpg was processed...
img/img-13.jpg was processed...
img/img-12.jpg was processed...
img/img-14.jpg was processed...
img/img-05.jpg was processed...
img/img-04.jpg was processed...
img/img-15.jpg was processed...
img/img-06.jpg was processed...
img/img-01.jpg was processed...
img/img-03.jpg was processed...
Finished in 11.153990578000048 seconds
```

Image Downloading Algorithm IN Serial:



```python
import requests, time

img_urls = [
    'https://unsplash.com/photos/ZEzHmf_PLiM/download?&force=true',
    'https://unsplash.com/photos/m9Kj085C00g/download?&force=true',
    'https://unsplash.com/photos/x8IaNrjRQHw/download?&force=true',
    'https://unsplash.com/photos/Px8b6Z-t0lk/download?&force=true',
    'https://unsplash.com/photos/SljSzLQTZNU/download?&force=true',
    'https://unsplash.com/photos/369TmUKDPt8/download?&force=true',
    'https://unsplash.com/photos/uARCs5MevTA/download?force=true',
    'https://unsplash.com/photos/v8XmVLHLJsI/download?&force=true',
    'https://unsplash.com/photos/YFlx4T220fg/download?&force=true',
    'https://unsplash.com/photos/W5XTTLpk1-I/download?&force=true',
    'https://unsplash.com/photos/GfFxvXdx_4w/download?&force=true',
    'https://unsplash.com/photos/iX0HSU4I_1g/download?&force=true',
    'https://unsplash.com/photos/TrTJelMQTNM/download?&force=true'
]

t1 = time.perf_counter()

for img_url in img_urls:
    img_bytes = requests.get(img_url).content
    img_name = img_url.split('/')[3]
    img_name = f'{img_name}.jpg'
    with open(img_name, 'wb') as img_file:
        img_file.write(img_bytes)
        print(f'{img_name} was downloaded...')

t2 = time.perf_counter()

print(f'Finished in {t2-t1} seconds')
```

## Output:

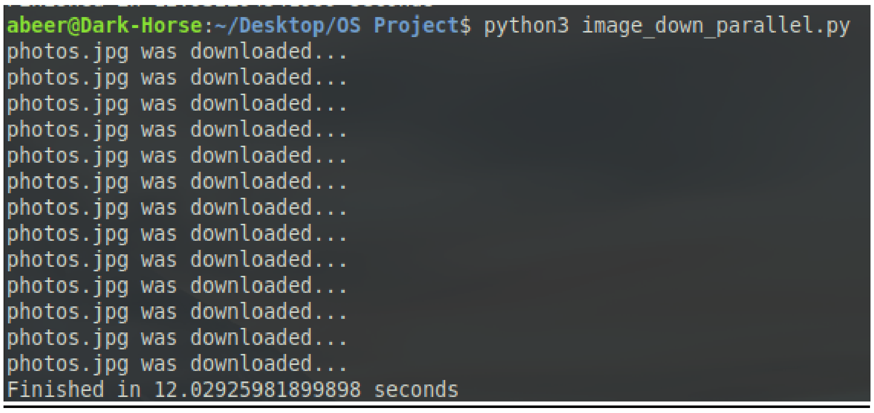


```
abeer@Dark-Horse:~/Desktop/OS Project$ python3 image_down_serial.py
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
Finished in 82.07415048799885 seconds
```

## Image Downloading Algorithm IN Parallel:

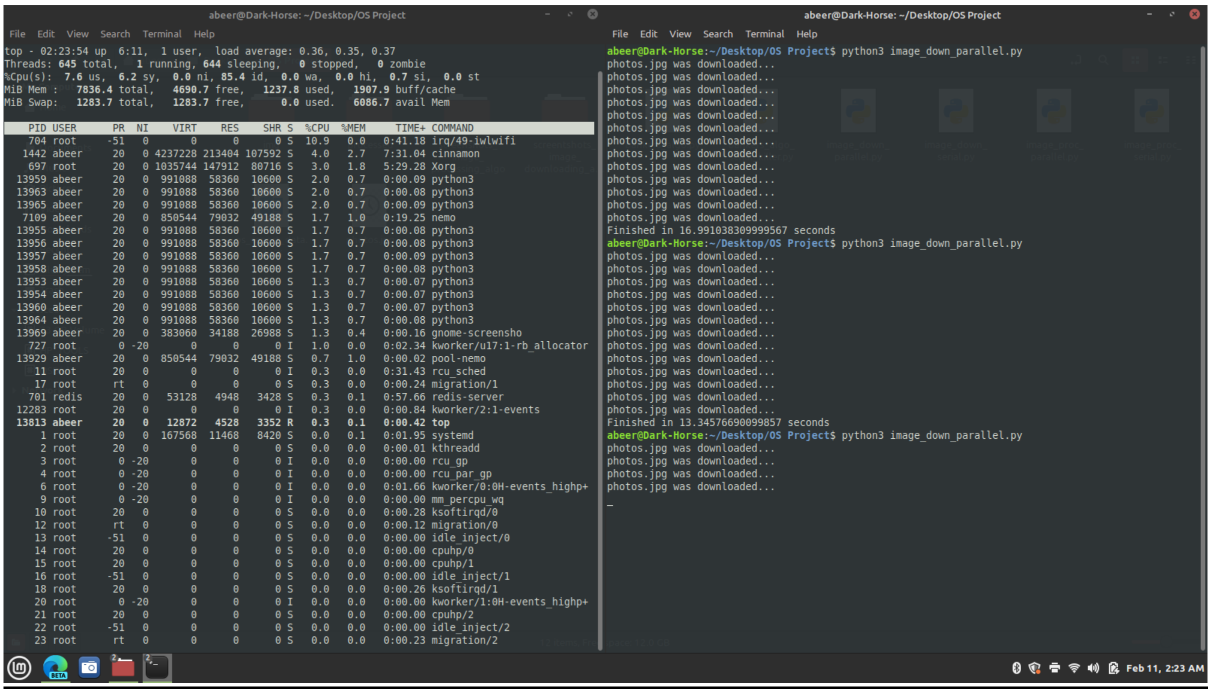
```python
import requests, time, threading

img_urls = [
    'https://unsplash.com/photos/ZEzHmf_PLiM/download?&force=true',
    'https://unsplash.com/photos/m9Kj085C00g/download?&force=true',
    'https://unsplash.com/photos/x8IaNrjRQHw/download?&force=true',
    'https://unsplash.com/photos/Px8b6Z-tOlk/download?&force=true',
    'https://unsplash.com/photos/SljSzLQTZNU/download?&force=true',
    'https://unsplash.com/photos/369TmUKDPt8/download?&force=true',
    'https://unsplash.com/photos/uARCs5MevTA/download?force=true',
    'https://unsplash.com/photos/v8XmVLHLJsI/download?&force=true',
    'https://unsplash.com/photos/YFlx4T220fg/download?&force=true',
    'https://unsplash.com/photos/W5XTTLpk1-I/download?&force=true',
    'https://unsplash.com/photos/GfFxvXdx_4w/download?&force=true',
    'https://unsplash.com/photos/iX0HSU4I_1g/download?&force=true',
    'https://unsplash.com/photos/TrTJelMQTNM/download?&force=true'
]
t1 = time.perf_counter()

def download_image(img_url):
    img_bytes = requests.get(img_url).content
    img_name = img_url.split('/')[3]
    img_name = f'{img_name}.jpg'
    with open(img_name, 'wb') as img_file:
        img_file.write(img_bytes)
        print(f'{img_name} was downloaded...')

threads = []

for img_url in img_urls:
    threads.append(threading.Thread(target=download_image, args=(img_url,)))

for thread in threads:
    thread.start()

for thread in threads:
    thread.join()

t2 = time.perf_counter()

print(f'Finished in {t2-t1} seconds')
```

## Output:



```
abeer@Dark-Horse:~/Desktop/OS Project$ python3 image_down_parallel.py
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
Finished in 12.02925981899898 seconds
```

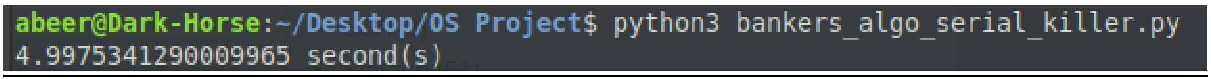
## Threads in Terminal:


abeer@Dark-Horse: ~/Desktop/OS Project
File Edit View Search Terminal Help
abeer@Dark-Horse: ~/Desktop/OS Project
File Edit View Search Terminal Help
Feb 11, 2:23 AM

```
top - 02:23:54 up  6:11,  1 user,  load average: 0.36, 0.35, 0.37
Threads: 645 total,   1 running, 644 sleeping,   0 stopped,   0 zombie
%Cpu(s):  7.6 us,  6.2 sy,  0.0 ni, 85.4 id,  0.0 wa,  0.0 hi,  0.7 si,  0.0 st
MiB Mem :   7836.4 total,   4690.7 free,   1237.8 used,   1907.9 buff/cache
MiB Swap:   1283.7 total,   1283.7 free,      0.0 used.   6086.7 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
    704 root     -51   0       0      0      0 S  10.9   0.0   0:41.18 irq/49-iwlwifi
   1442 abeer     20   0 4237228 213404 107592 S   4.0   2.7   7:31.04 cinnamon
    697 root      20   0 1035744 147912  80716 S   3.0   1.8   5:29.28 Xorg
  13959 abeer     20   0  991088  58360  10600 S   2.0   0.7   0:00.09 python3
  13963 abeer     20   0  991088  58360  10600 S   2.0   0.7   0:00.08 python3
  13965 abeer     20   0  991088  58360  10600 S   2.0   0.7   0:00.09 python3
   7109 abeer     20   0  850544  79032  49188 S   1.7   1.0   0:19.25 nemo
  13955 abeer     20   0  991088  58360  10600 S   1.7   0.7   0:00.08 python3
  13956 abeer     20   0  991088  58360  10600 S   1.7   0.7   0:00.08 python3
  13957 abeer     20   0  991088  58360  10600 S   1.7   0.7   0:00.09 python3
  13958 abeer     20   0  991088  58360  10600 S   1.7   0.7   0:00.08 python3
  13953 abeer     20   0  991088  58360  10600 S   1.3   0.7   0:00.07 python3
  13954 abeer     20   0  991088  58360  10600 S   1.3   0.7   0:00.07 python3
  13960 abeer     20   0  991088  58360  10600 S   1.3   0.7   0:00.07 python3
  13964 abeer     20   0  991088  58360  10600 S   1.3   0.7   0:00.08 python3
  13969 abeer     20   0  383060  34188  26988 S   1.3   0.4   0:00.16 gnome-screensho
    727 root       0 -20       0      0      0 I   1.0   0.0   0:02.34 kworker/u17:1-rb_allocator
  13929 abeer     20   0  850544  79032  49188 S   0.7   1.0   0:00.02 pool-nemo
     11 root      20   0       0      0      0 I   0.3   0.0   0:31.43 rcu_sched
     17 root      rt   0       0      0      0 S   0.3   0.0   0:00.24 migration/1
    701 redis     20   0   53128   4948   3428 S   0.3   0.1   0:57.66 redis-server
  12283 root      20   0       0      0      0 I   0.3   0.0   0:00.84 kworker/2:1-events
  13813 abeer     20   0   12872   4528   3352 R   0.3   0.1   0:00.42 top
      1 root      20   0  167568  11468   8420 S   0.0   0.1   0:01.95 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:00.01 kthreadd
      3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_gp
      4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_par_gp
      6 root       0 -20       0      0      0 I   0.0   0.0   0:01.66 kworker/0:0H-events_highp+
      9 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 mm_percpu_wq
     10 root      20   0       0      0      0 S   0.0   0.0   0:00.28 ksoftirqd/0
     12 root      rt   0       0      0      0 S   0.0   0.0   0:00.12 migration/0
     13 root     -51   0       0      0      0 S   0.0   0.0   0:00.00 idle_inject/0
     14 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/0
     15 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/1
     16 root     -51   0       0      0      0 S   0.0   0.0   0:00.00 idle_inject/1
     18 root      20   0       0      0      0 S   0.0   0.0   0:00.26 ksoftirqd/1
     20 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/1:0H-events_highp+
     21 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/2
     22 root     -51   0       0      0      0 S   0.0   0.0   0:00.00 idle_inject/2
     23 root      rt   0       0      0      0 S   0.0   0.0   0:00.23 migration/2
```

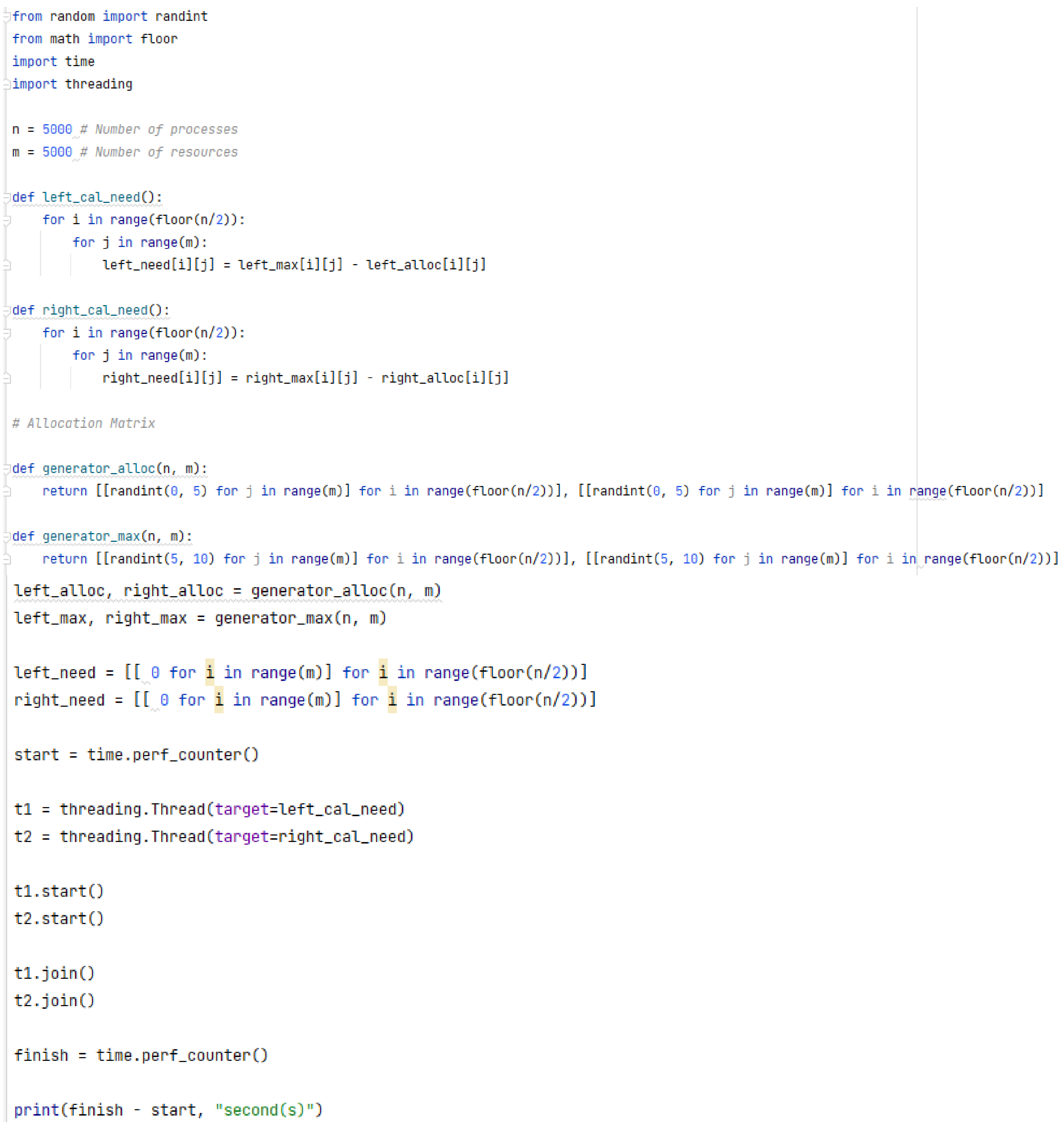
```
abeer@Dark-Horse:~/Desktop/OS Project$ python3 image_down_parallel.py
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
Finished in 16.991038309999567 seconds
abeer@Dark-Horse:~/Desktop/OS Project$ python3 image_down_parallel.py
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
Finished in 13.34576690099857 seconds
abeer@Dark-Horse:~/Desktop/OS Project$ python3 image_down_parallel.py
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
photos.jpg was downloaded...
_
```

## In Banker’s Algorithm for serial computing we got total time:



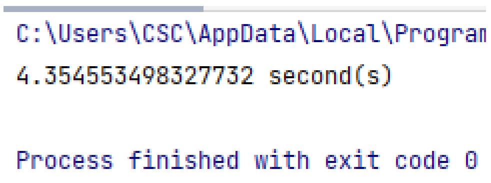
```
abeer@Dark-Horse:~/Desktop/OS Project$ python3 bankers_algo_serial_killer.py
4.9975341290009965 second(s)
```

## Banker's Algorithm IN Parallel:



```
from random import randint
from math import floor
import time
import threading

n = 5000 # Number of processes
m = 5000 # Number of resources

def left_cal_need():
    for i in range(floor(n/2)):
        for j in range(m):
            left_need[i][j] = left_max[i][j] - left_alloc[i][j]

def right_cal_need():
    for i in range(floor(n/2)):
        for j in range(m):
            right_need[i][j] = right_max[i][j] - right_alloc[i][j]

# Allocation Matrix

def generator_alloc(n, m):
    return [[randint(0, 5) for j in range(m)] for i in range(floor(n/2))], [[randint(0, 5) for j in range(m)] for i in range(floor(n/2))]

def generator_max(n, m):
    return [[randint(5, 10) for j in range(m)] for i in range(floor(n/2))], [[randint(5, 10) for j in range(m)] for i in range(floor(n/2))]
left_alloc, right_alloc = generator_alloc(n, m)
left_max, right_max = generator_max(n, m)

left_need = [[ 0 for i in range(m)] for i in range(floor(n/2))]
right_need = [[ 0 for i in range(m)] for i in range(floor(n/2))]

start = time.perf_counter()

t1 = threading.Thread(target=left_cal_need)
t2 = threading.Thread(target=right_cal_need)

t1.start()
t2.start()

t1.join()
t2.join()

finish = time.perf_counter()

print(finish - start, "second(s)")
```

Output:



```
C:\Users\CSC\AppData\Local\Progran
4.354553498327732 second(s)

Process finished with exit code 0
```

## 4. Conclusion

As the major drawback with the serial computing was the one by one execution of the instructions which were taking an abundance of time when it comes to heavy I/O or CPU bound tasks. The main purpose of this report was to perform comparative analysis on serial vs parallel computing approaches, and deduce a scenario in which an algorithm is highly efficient and takes notably less computing time.
The investigation of this report was based on two algorithms - Image processing and Image downloading. In the case of the image processing algorithm, one by one execution of the instructions were taking a lot of the time to execute, whereas the results were significantly improved by running them parallely. This is the same case in the image downloading algorithm which uses multiple threads to execute its instructions concurrently.
But, there are some cases where parallel execution of the instructions do not affect the computing time of a program, even in some cases overhead in the computation time is observed, resulting in no performance gain.
Therefore, this report concludes that independent processes can be run parallely to improve overall execution time of a program.