DSAI 3202 – Parallel and distributed computing
Lab – 3 Part 1: Data Parallel Model

1. Objectives:

- Build a data parallel model program using threads in Python.
- Build a data parallel model program using processes in Python.
- Understand the basics of parallel programming using Python's **`threading`** and **`multiprocessing`** modules.

2. Hints for this lab:

2.a. Creating thread using a for loop

This code is generic!

```
1.  import threading
2.
3.  def worker(thread_id):
4.      print(f"Thread {thread_id} started")
5.      # Add your thread's code here
6.      print(f"Thread {thread_id} finished")
7.
8.  # Number of threads to create
9.  num_threads = 4
10.
11. # Create and start threads in a loop
12. threads = []
13. for i in range(num_threads):
14.     thread = threading.Thread(target=worker, args=(i,))
15.     threads.append(thread)
16.     thread.start()
17.
18. # Wait for all threads to finish
19. for thread in threads:
20.     thread.join()
21.
22. print("All threads have finished")
```

2.b. Creating processes using a for loop

This code is generic!

```
1.  import multiprocessing
2.
3.  def worker(process_id):
4.      print(f"Process {process_id} started")
5.      # Add your process's code here
6.      print(f"Process {process_id} finished")
7.
8.  # Number of processes to create
9.  num_processes = 4
```

```
10.
11.   # Create and start processes in a loop
12.   processes = []
13.   for i in range(num_processes):
14.       process = multiprocessing.Process(target=worker, args=(i,))
15.       processes.append(process)
16.       process.start()
17.
18.   # Wait for all processes to finish
19.   for process in processes:
20.       process.join()

21.   print("All processes have finished")
```

3. Tasks

3.a. The sequential Case

a. Write a Python program that calculates the sum of all numbers from 1 to a given large number **n**.
b. Measure the execution time of the program using the time module.
c. Print the sum and the execution time (Put them in variables).

3.b. Parallelize with Threading:

a. Modify your program to use the **`threading`** module to parallelize the summation.
b. Divide the range of numbers (1 to **n**) into multiple equal parts and assign each part to a separate thread *(Hint: make sure to make a copy of each part)*.
c. Each thread should calculate the sum of its assigned range.
d. Measure the execution time and compare it with the sequential version.
e. Print the sum and execution time.

3.c. Parallelize with Multiprocessing

f. Modify your program to use the **`multiprocessing`** module to parallelize the summation.
g. Divide the range of numbers (1 to **n**) into multiple equal parts, and assign each part to a separate process.
h. Each process should calculate the sum of its assigned range.
i. Measure the execution time and compare it with the sequential and threaded versions.

3.d. Questions:

a. How does the execution time change when moving from sequential to threaded to multiprocessing implementations?
b. For each case, compute:
    i. *The speedup,*
    ii. *The efficiency,*
    iii. *The speedups using Amdhal's Law,*
    iv. *The speedups Gustaffson's Law.*
c. Are there any performance differences between the threaded and multiprocessing versions?
d. What challenges did you face when implementing parallelism, and how did you address them?

e. When would you choose threading over multiprocessing or vice versa for parallel tasks?