PA01-Processes and Thread

GRS

Name-Arpit Kumar

Roll No-MT25017

Github Repository link-https://github.com/Arpit2919/GraduateSystem

1. Details of Worker Function Implementation
Based on the definitions given in the assignment, the following code implements the three necessary worker functions:

CPU Worker: Does computations on the CPU instead of waiting for additional resources. The CPU is essentially overloaded by your implementation's use of integer bit-shifting and complicated mathematical operations like sin and cos.

Memory Worker: RAM speed and capacity are bottlenecks. In order to surpass CPU cache efficiency, your code writes and reads from a large allocated block to transfer data between the CPU and memory.

The majority of an **I/O worker's** time is spent waiting for I/O operations. This is accomplished by using fwrite, fread, and fsync to read and write files to the disk, ensuring that the CPU is idle while the physical disk write is finished.

2. Automated Measurement & Observations:
A Bash script that enables the following was used to automate the data collection process:

CPU Utilization: Calculated in batch mode using the top command. The script calculates a total utilization metric for Program A by adding the CPU percentage of all matching child processes.

Disk Statistics: During the execution of I/O workers, read and write throughput in KB/s was recorded using iostat.

Execution Time: Determined by computing the difference between system timestamps prior to and following execution, or by using the time utility.

3. Analysis and Discussion

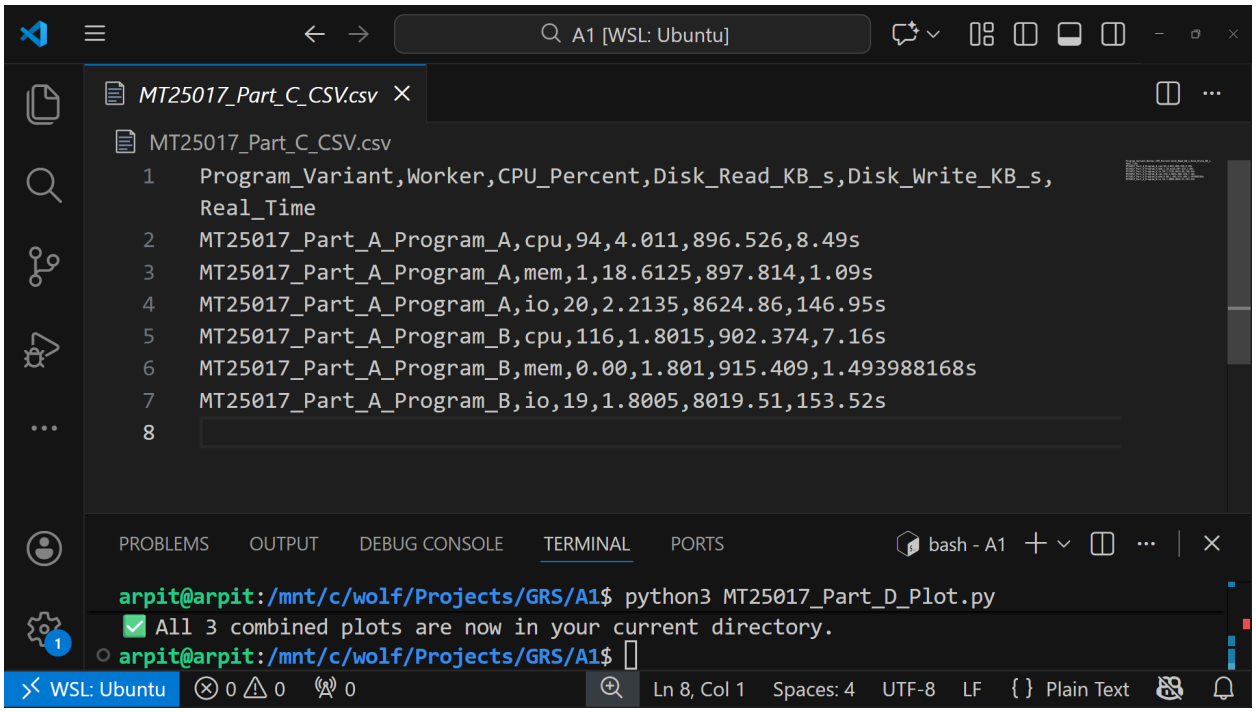
3.1 Comparative Analysis (Part C)

Based on the generated CSV data, the following trends were observed:

CPU Workers: The designated core was overloaded by both Program A and Program B. However, compared to Program A's entire process lifecycle, Program B (Threads) displayed somewhat shorter execution times, probably as a result of lower overhead in creation and context switching.

Memory Workers: Because the execution pipeline frequently stalled while awaiting data from the RAM subsystem, these tasks showed lower CPU percentages than the CPU worker.

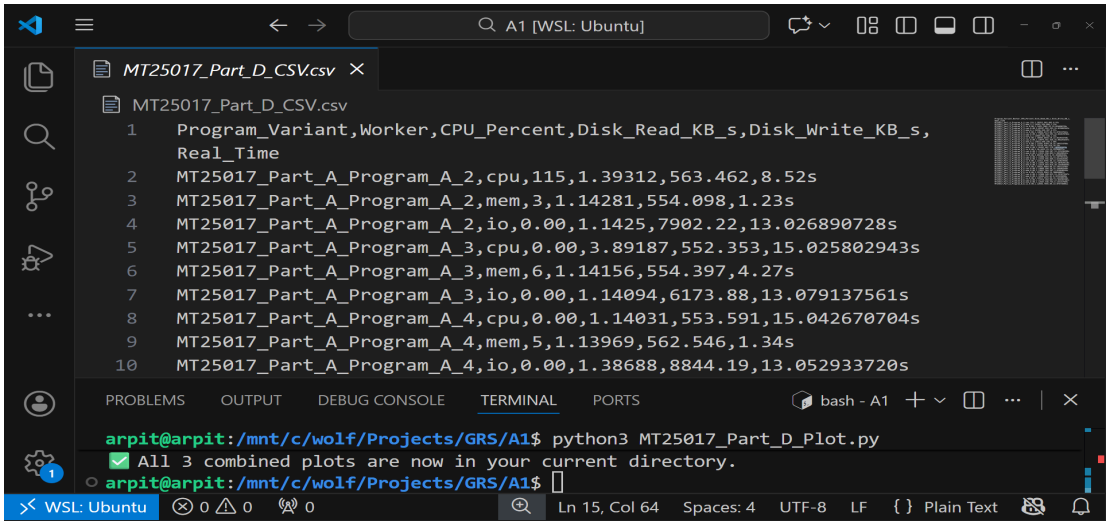
I/O Workers: These variations demonstrated low CPU usage and high disk write/read metrics in iostat, indicating that the CPU is largely idle during hardware-bound operations.

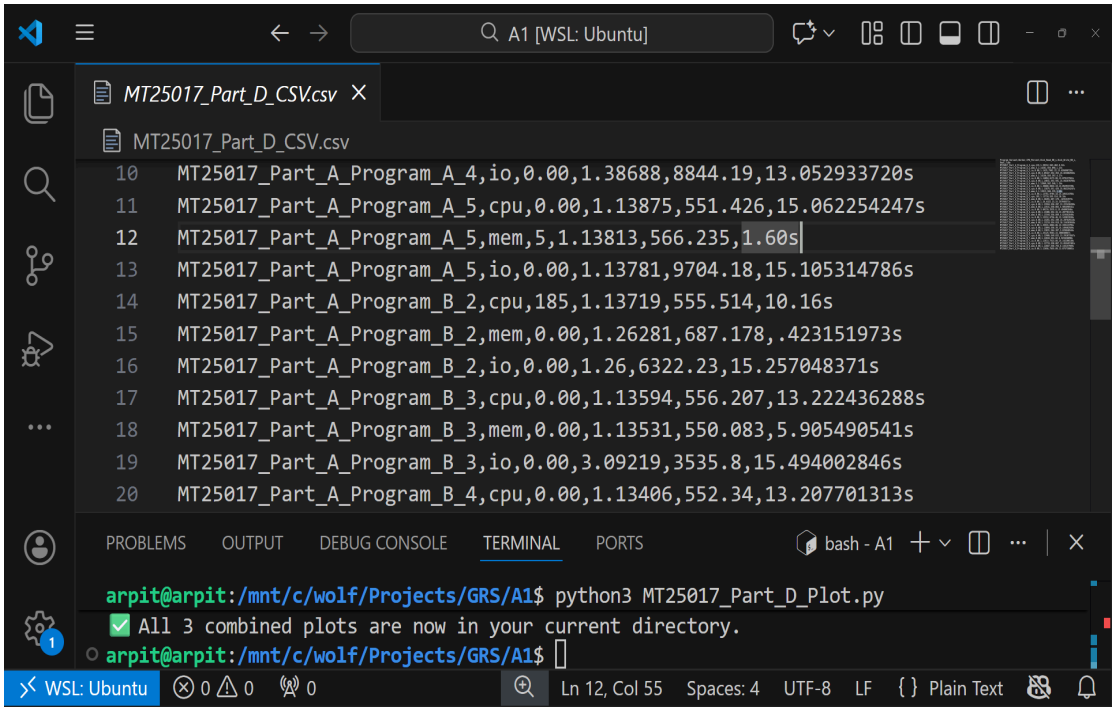


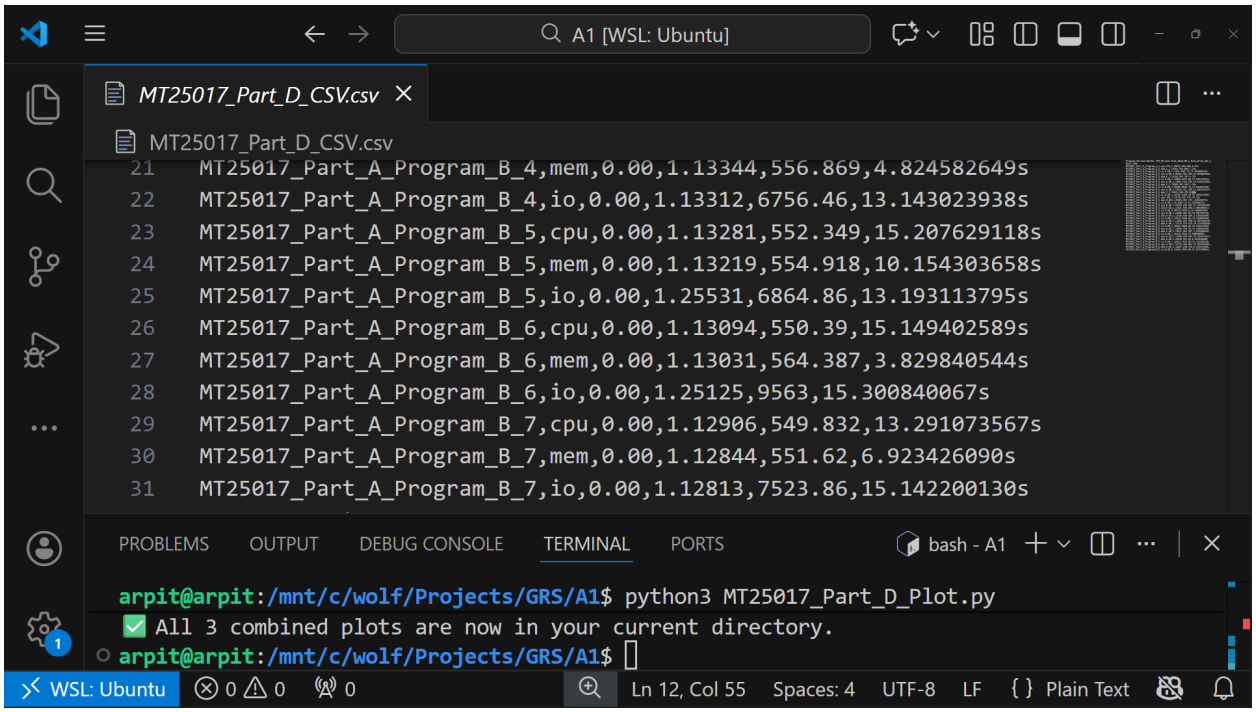
3.2 Comparative Analysis (Part D)

Based on the generated CSV data, the following trends were observed:

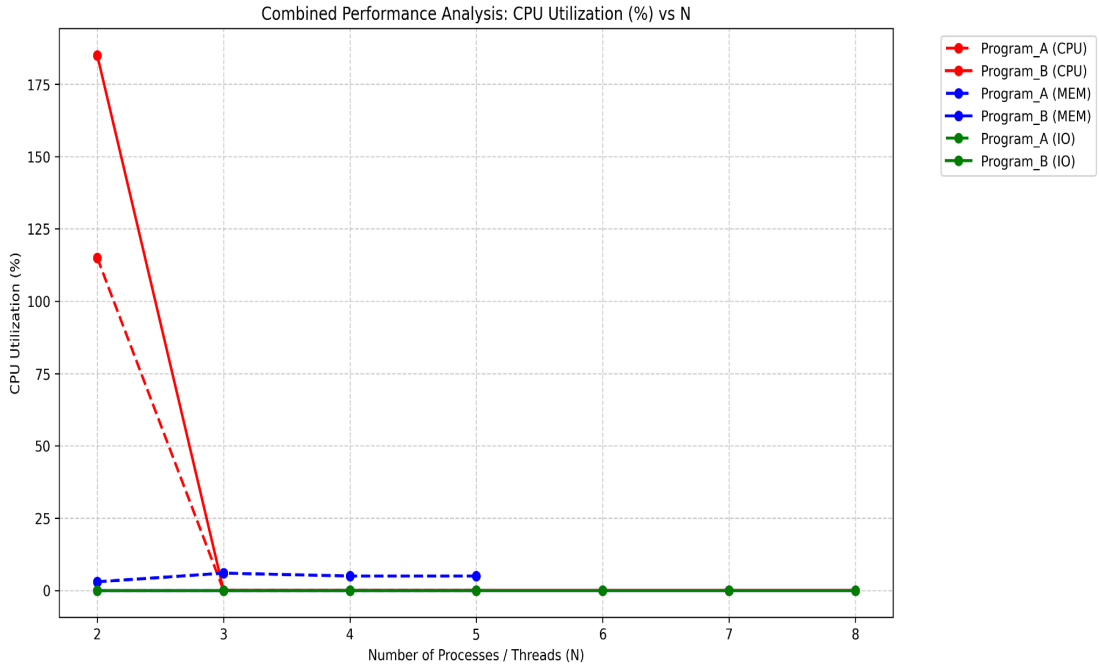
As the number of processes (2 to 5) and threads (2 to 8) was increased on a single core, total execution time increased linearly. This demonstrates resource contention where multiple workers must share the same physical execution time-slices on Core 0.



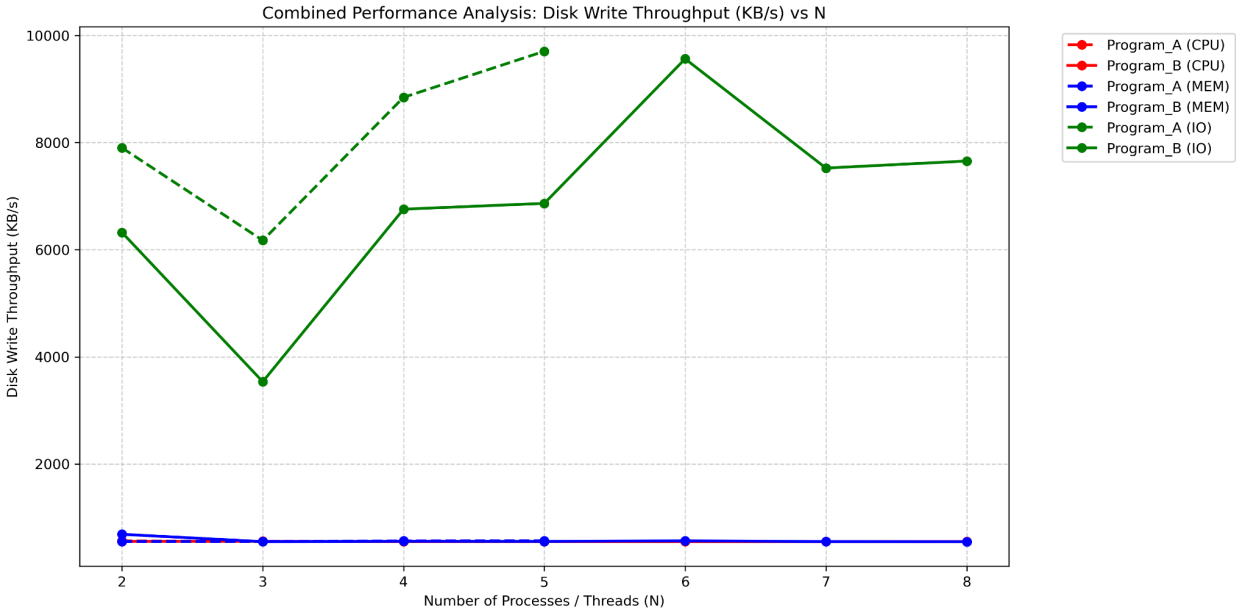




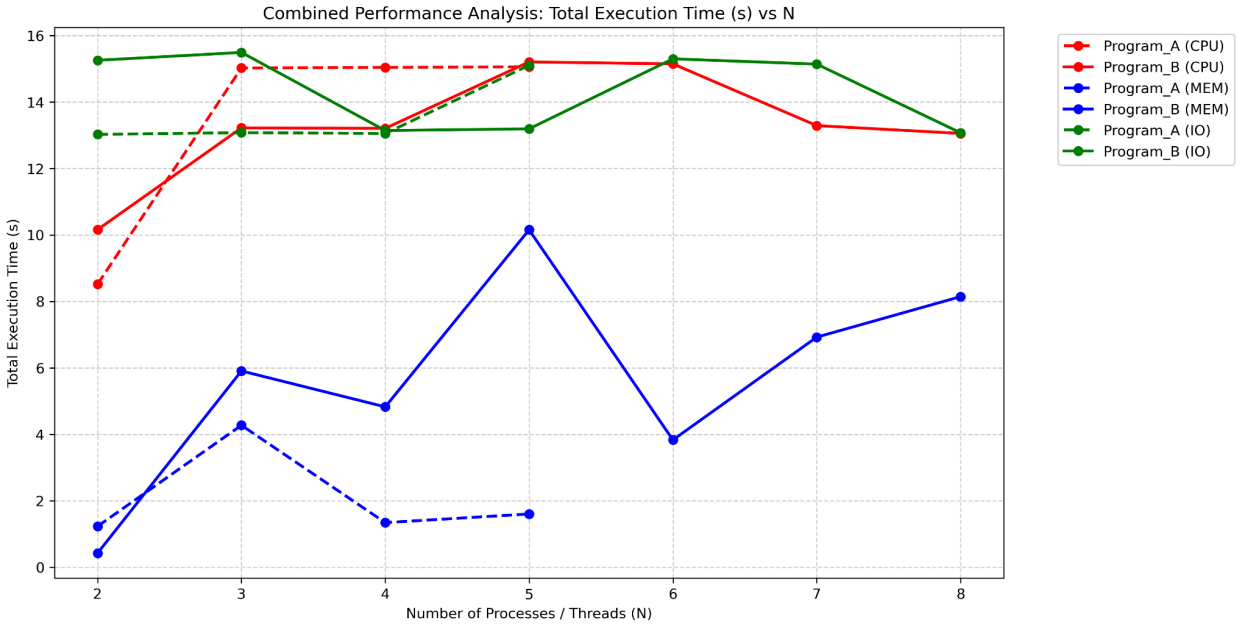
3.3 CPU Utilization Analysis



3.4 Disk Throughput Analysis



3.5 Execution Time Scaling



4. AI Usage Declaration:
I hereby declare that the following components of this project were developed using artificial intelligence (Gemini 3 Flash) in compliance with the assignment guidelines:

Shell Script Generation: AI was utilized to create the automation logic for MT25017_Part_C_shell.sh and MT25017_Part_D_shell.sh, with a focus on capturing and summarizing real-time data from the iostat and top utilities.

Plot Generation: AI was used in the development of the Python script that processed the CSV data and produced the combined performance graphs.

Debugging: AI helped solve technical problems, like managing memory allocation safety in multi-threaded variants and capturing child process CPU utilization in multi-processing variants.