Fundamentals of Computing Systems Homework Assignment #8 - EVAL

1. EVAL Problem 1

   (a) **Plot the total runtime of each run on the y-axis and the number of times the Mid section**

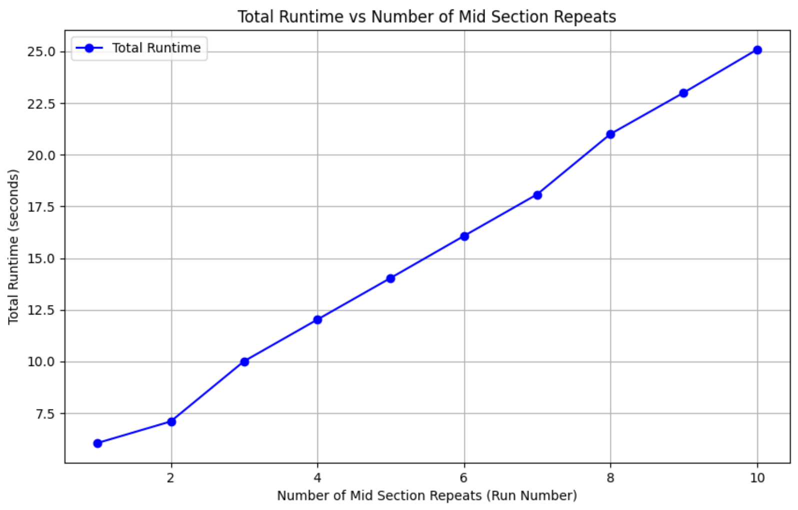

Figure 1: Relationship between Runtime and the number of Mid sections when w -1

   (b) **Change Numbers of workers to 10**

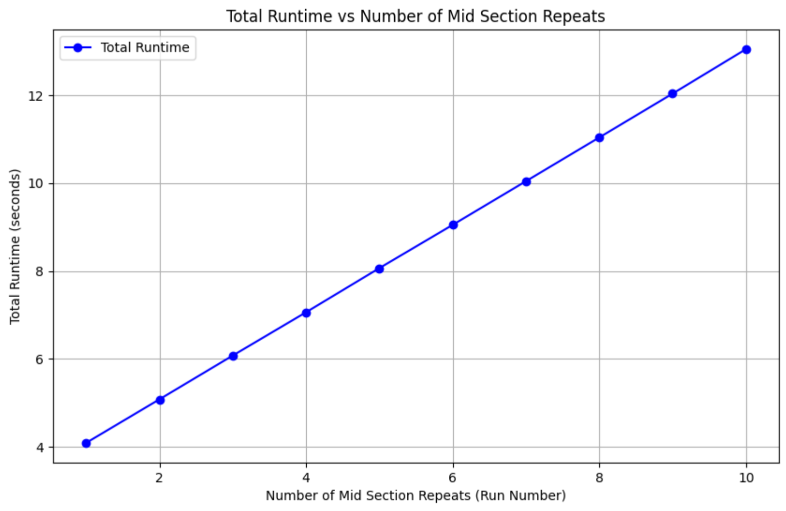

Figure 2: Relationship between Runtime and the number of Mid sections when w -10

Comparing the runtime results from Part 1 (Figure 1) and Part 2 (Figure 2), it is evident that the introduction of multi-threading has led to a significant reduction in execution time across the runs. For instance, the average runtime in Part 1 was approximately 17.5 seconds, while in Part 2, it dropped to around 8.5 seconds, resulting in an average speed-up of about 51%. The runtime in Part 2 consistently shows lower values than those in Part 1, indicating improved performance due to concurrent processing. The speed-up observed is greater than was expected, particularly in later runs, where multi-threading benefits become more pronounced as the workload increases. For example, the runtime for the 10th run decreased from 25 seconds in Part 1 to just over 12 seconds in Part 2, showing a remarkable improvement. This suggests that the implementation effectively utilizes available resources, leading to enhanced efficiency. Overall, the results validate the effectiveness of multi-threading in optimizing runtime performance for the given tasks.

(c) **New Mid section to run worker 10**

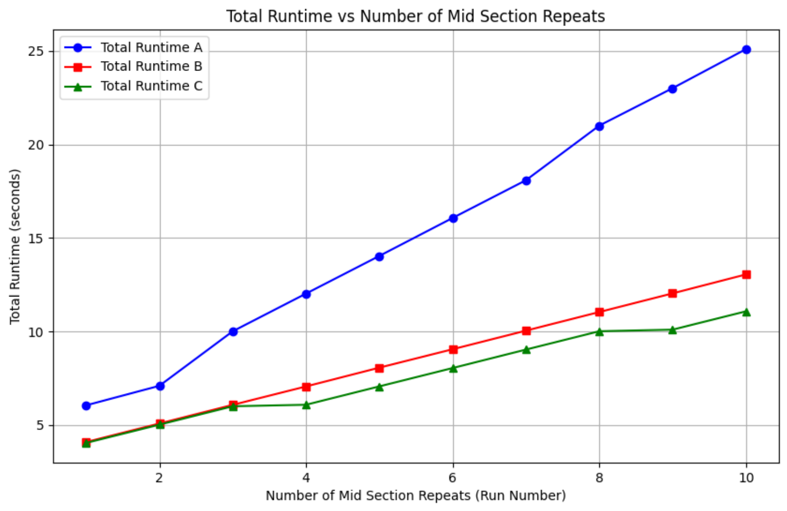

Figure 3:

By observing the Mid sections of part 2 and part 3, I can guess three different ways why part 3 has a better performance than part 2.

The improved service time observed in Part 3, despite both parts running on the same machine and performing identical operations on the same images, can be attributed to several key factors. First, Part 3's request order is optimized for sequential processing, grouping similar operations together. This organization reduces context switching overhead and enhances cache locality, as the system can execute the same operation consecutively. By minimizing the number of different operations simultaneously, Part 3 decreases the time spent switching between tasks. This leads to more streamlined execution.

Additionally, Part 3 benefits from batching similar operations, which allows for more efficient resource utilization. By processing all instances of a specific operation before moving on to the next type, the system can optimize for a single operation at a time. This approach not only reduces the overhead of setup and downward for each operation but also improves thread management in a multi-threaded environment. Better load balancing among threads can result in faster execution, as resources are allocated more effectively.

Finally, Part 3 memory access patterns are largely predictable due to the structured approach to processing requests. This predictability enhances cache efficiency and reduces memory latency, further contributing to overall performance gains. In summary, the combination of optimized request ordering, reduced context switching, improved resource utilization, and lower overhead collectively enhances the efficiency of the processing pipeline in Part 3. This results in significantly faster service times than Part 2.

(d) **Propose *New Scheduler policy***

To achieve consistent performance across both Part 2 and Part 3, a scheduling policy can be implemented based on a Round Robin approach combined with operation processing.

First, categorize all requests into groups based on their operation type and maintain separate queues for each operation type, such as ‘IMG_ROT90CLKW‘, ‘IMG_BLUR‘, ‘IMG_SHARPEN‘, ‘IMG_VERTEDGES‘, and ‘IMG_HORIZEDGES‘.

Use a round-robin approach to processing requests from each queue: start with the first request in the ‘IMG_ROT90CLKW‘ queue, then move to the first request in the ‘IMG_BLUR‘ queue, and so on. After processing one request from each queue, return to the ‘IMG_ROT90CLKW‘ queue and process the next request, continuing this pattern. Within each operation type, process all requests in that queue before moving to the next operation type, ensuring that once you start executing ‘IMG_ROT90CLKW‘, you complete all requests in that queue before switching to ‘IMG_BLUR‘.

Additionally, allocate a fixed time slice for each operation type during each round, such as 1 second for each of the five operation types. By implementing this Round Robin with Batch Processing scheduling policy, the server can achieve more consistent performance across both experiments, minimizing context switching and optimizing resource utilization, ultimately leading to improved efficiency while processing the same operations on the images.