Assumption:

I tested directory **5** for extra analysis because it has the most number of images to run.

In Makefile, we revised line 18 to "**/usr/bin/time -f "%e %P %M" ./image_rotation img/$$dir_num output/$$dir_num 10 180;**\" for testing relevant metrics.

In the following graphs:

1. **# of Threads**: number of worker threads changed by Makefile
2. **Elapsed Time(s)** [%e]: Elapsed real (wall clock) time used by the process, in seconds.
3. **CPU(%)** [%P]: Percentage of the CPU that this job got. This is just user + system times divided by the total running time. It also prints a percentage sign.
4. **MaxSetSize Memory(KB)** [%M]: Maximum resident set size of the process during its lifetime, in Kilobytes.

Figures:

# of Threads	Elapsed Time(s)	CPU(%)	MaxSetSize Memory(KB)
1	0.12	4	2528
10	0.1	9	3648
30	0.11	8	4284
50	0.1	13	4284
80	0.1	21	4796
100	0.11	12	4896
120	0.12	19	4604
150	0.11	22	5948
180	0.11	16	5600
200	0.11	15	5820
250	0.11	18	6076
300	0.13	22	7348

Figure 1: Overall Results of Directory 5 in Average

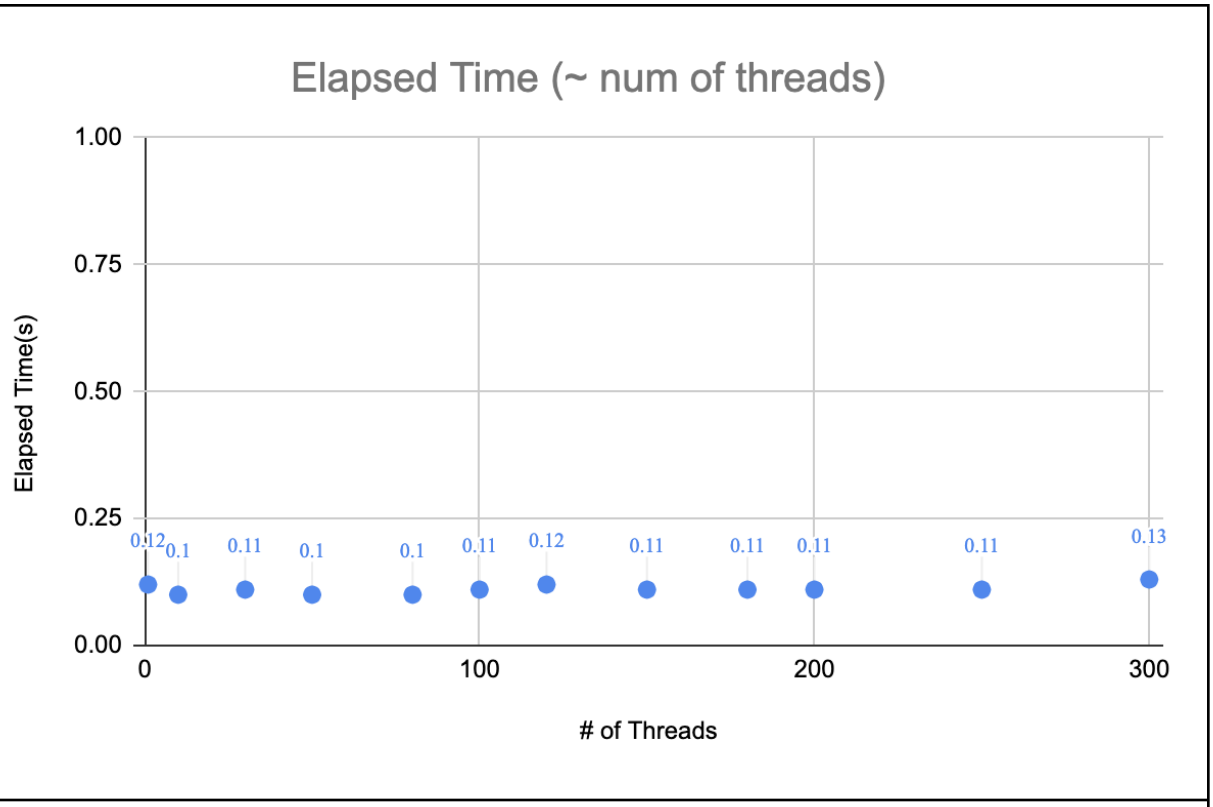

Figure 2: Elapsed Time(s)

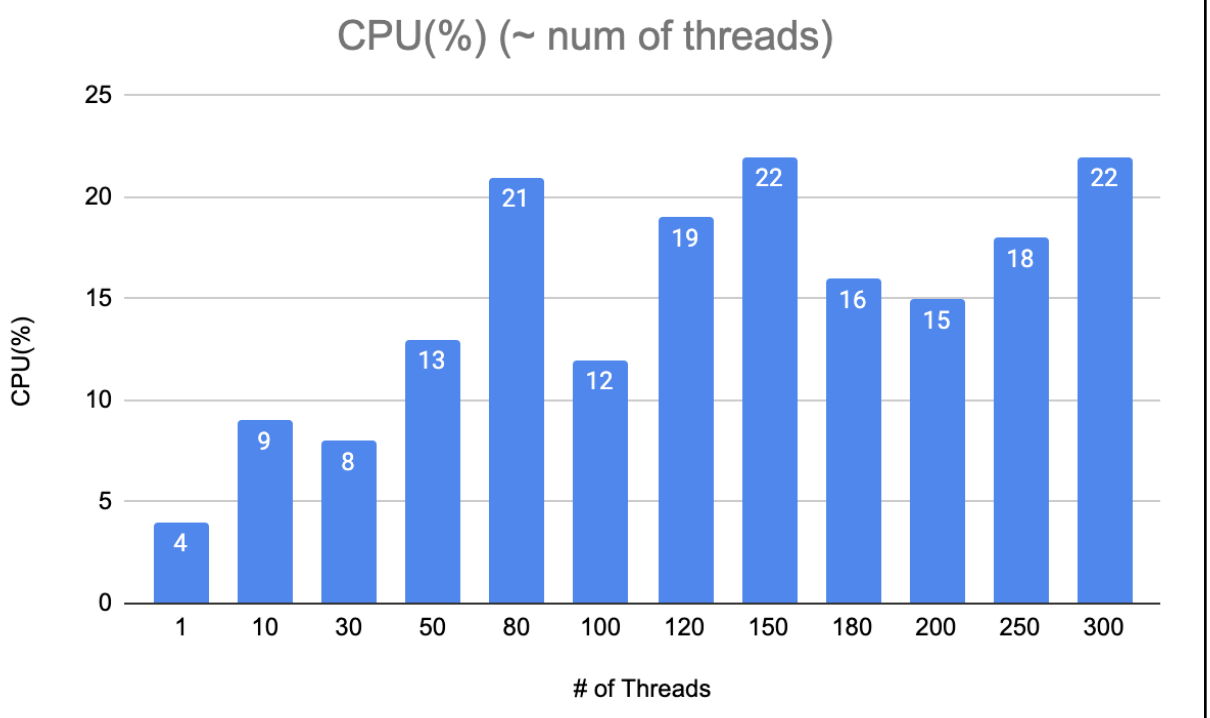

Figure 3: CPU(%)

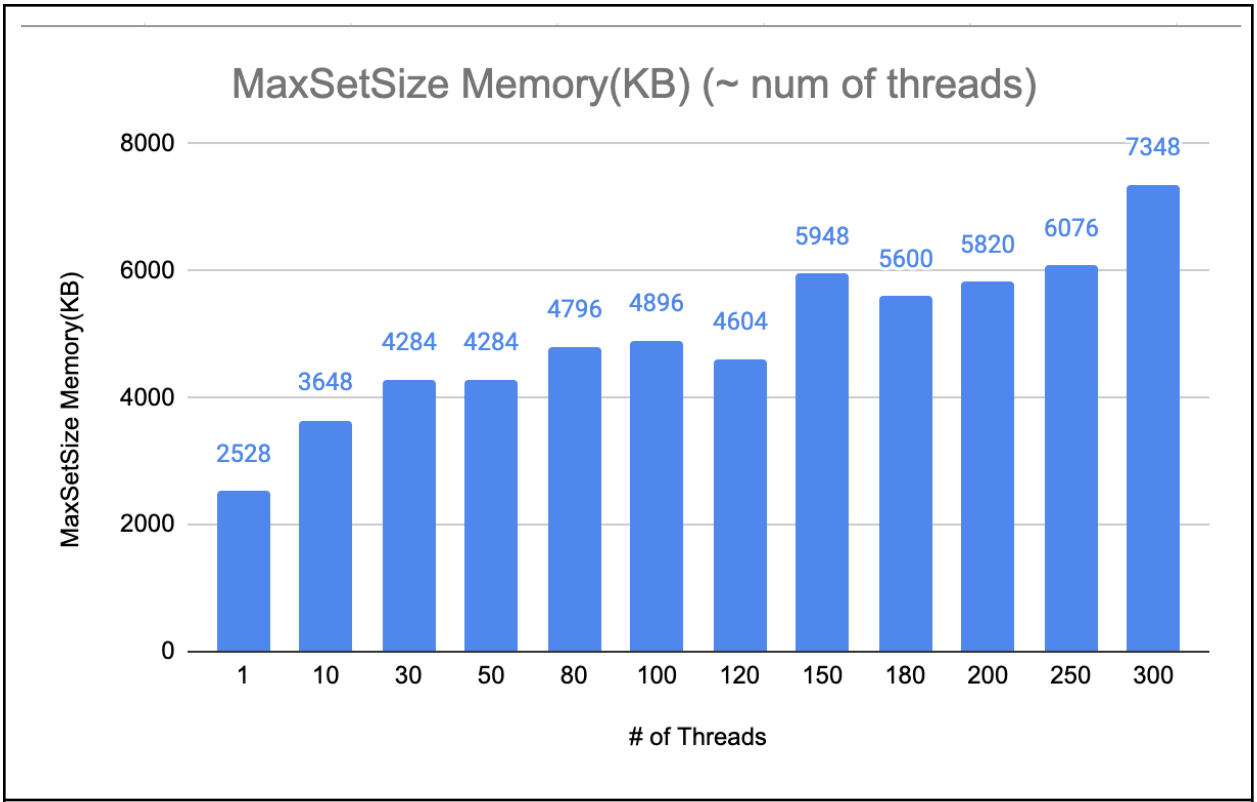

Figure 4: MaxSetSize Memory(KB)

Analysis:

Elapsed Time:

This graph indicates that the time it takes to complete the image rotation task remains fairly consistent as the number of threads increases, only with a slight increase in elapsed time when reaching 300 threads. This suggests that the task scales well up to a certain point. This may be due to only 19 images processed and the uncomplicated Image processing methods. As the number of threads increases, and due to limitations of hardware, the number and time of image worker threads are always fixed, so the time difference is small. But when the thread is 1, the time will be slightly longer than the later one, because the number of threads is smaller than the number of images.

CPU Usage:

CPU utilization shows an increasing trend as the number of threads increases. Starting at 4% for a single thread, it peaks at around 21-22% for 80 and 300 threads. This could indicate that the CPU is able to handle more threads without becoming a bottleneck, but there is a limit to how effectively additional threads can be used before they just add overhead without improving hardware performance.

Memory Usage:

Memory usage increases as the number of threads increases, which is expected because each thread will have its own stack and possibly other data structures allocated in memory. Memory usage does not increase linearly, which may suggest some efficiency in how the program handles memory as more threads are added, but there is still a clear upward trend.

Suggestions:

Based on the data, there may be an optimal range of thread counts where the performance (in terms of execution time and resource usage) is maximized. It would be important to consider other factors such as the nature of the workload, the capabilities of the hardware, and the efficiency of the thread management in the operating system. For optimizing the multi-threaded image rotation program, I recommend using a thread count that balances CPU and memory usage without causing significant increases in execution time. The data suggests that this might be around the range of 50 to 80 threads.

Conclusion:

The project seems to scale well with an increasing number of threads up to a point. However, there seems to be a threshold beyond which adding more threads does not result in performance improvement. This could be due to several factors, such as locking mechanisms, or a limitation in the CPU's ability to handle more threads. However, there are great limitations in the analysis of this project. For example, the number of image processing is not large and complex enough. It cannot give us an accurate analysis, but it can be used for our reference.