UNIVERSITÀ
DEGLI STUDI
FIRENZE

Faculty of Engineering

Master course in
Artificial Intelligence

PERFORMANCE ANALYSIS REPORT FOR IMAGE READING

Parallel Programming for Machine Learning

Mid-term assignment

Candidate
Loric TONGO GUIMTSA

Professor
Marco BERTINI
Associate Professor

Academic Year: 2023/2024

Contents

1) Introduction 3
2) Context 3
3) Methodology 3
4) Sequential vs Parallel 3
   a) Sequential Program 3
   b) Parallel Program 4
5) Results 5
   a) Sequential Program 5
   b) Parallel Program 5
6) Performance Analysis 8
7) Conclusion 8
8) Recommendations 9

1) Introduction

This report presents a comparative performance analysis between a sequential image reader program and a parallel image reader program implemented using threads.

2) Context

The subject concerns the development of a multithreaded JPEG image reader that can read a certain number of images from a given directory and store the uncompressed images or the compressed stream in an appropriate data structure. The goal is to evaluate the performance of both sequential and parallel approaches.

3) Methodology

We implemented two programs to read the images. The first is a sequential program that reads the images one by one, while the second is a parallel program that utilizes threads to read the images simultaneously. The performance of each program was measured in terms of execution time and processing speed.

4) Sequential vs Parallel

a) Sequential Program

The sequential program reads each image in the specified directory and processes them one after the other. The total execution time and reading speed are measured. The program is simple and easy to understand, but it does not leverage multiple CPU cores.

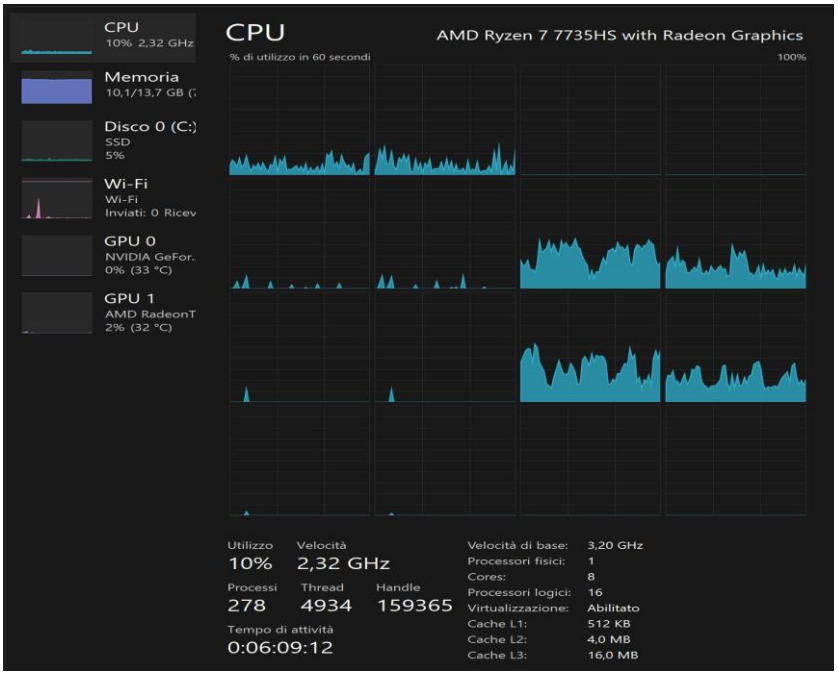

Figure 1: CPU utilization

As evidenced by Figure 1, which depicts CPU utilization during the execution of the sequential program, it's apparent that the sequential program does not utilize all CPU cores. The figure clearly illustrates that only about 10% of the CPU power is utilized during the execution of the sequential program.

b) Parallel Program

The parallel program uses threads to read the images in parallel. It divides the images into multiple tasks, with each task being processed by a separate thread. The number of threads equals the number of logical CPU cores. The total execution time and reading speed are also measured for this approach.

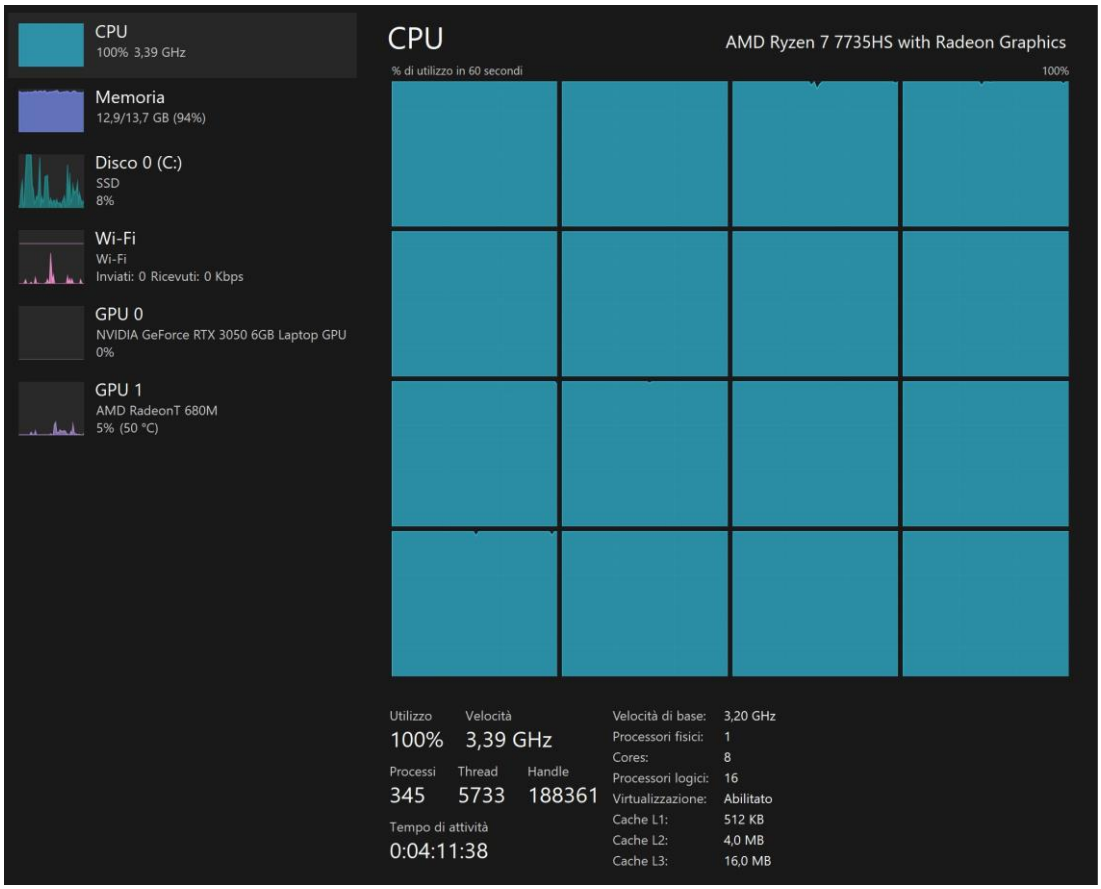

Figure 2: CPU utilization

As illustrated in Figure 2, which depicts CPU utilization during the execution of the parallel program, it's evident that the parallel program utilizes all CPU cores. The figure clearly shows that the parallel program efficiently utilizes the entire processing power of the CPU.

5) Results

To compute the speedup, we will use this formulation:
Speedup = Sequential execution time / Parallel execution time

a) Sequential Program

- Number of images: 2.000
  - Execution time: 12 second*
  - Processing speed: 168 images/second*

- Number of images: 5.000
  - Execution time: 28 second*
  - Processing speed: 181 images/second*

- Number of images: 7.000
  - Execution time: 39 second*
  - Processing speed: 180 images/second*

- Number of images: 10.000
  - Execution time: 56 second*
  - Processing speed: 178 images/second*

- Number of images: 15.000
  - Execution time: 79 second*
  - Processing speed: 190 images/second*

- Number of images: 200.000
  - Execution time: 728 second*
  - Processing speed: 275 images/second*

b) Parallel Program

- Using 2 processes:
  - Number of images: 2.000
    - Execution time: 6 seconds
    - Processing speed: 305 images/second
    - **Speedup** = 12 / 6 = 2

  - Number of images: 5.000
    - Execution time: 14 seconds
    - Processing speed: 354 images/second
    - **Speedup** = 28 / 14 = 2

- Number of images: 7.000
  - Execution time: 18 seconds
  - Processing speed: 376 images/second
  - **Speedup** = 39 / 18 = 2.16

- Number of images: 10.000
  - Execution time: 29 seconds
  - Processing speed: 347 images/second
  - **Speedup** = 56 / 29 = 1.93

- Number of images: 15.000
  - Execution time: 42 seconds
  - Processing speed: 356 images/second
  - **Speedup** = 79 / 42 = 1.88

➢ Using 5 processes:

- Number of images: 2.000
  - Execution time: 3 seconds
  - Processing speed: 629 images/second
  - **Speedup** = 12 / 3 = 4

- Number of images: 5.000
  - Execution time: 6 seconds
  - Processing speed: 800 images/second
  - **Speedup** = 28 / 6 = 4.66

- Number of images: 7.000
  - Execution time: 9 seconds
  - Processing speed: 813 images/second
  - **Speedup** = 39 / 9 = 4.33

- Number of images: 10.000
  - Execution time: 13 seconds
  - Processing speed: 788 images/second
  - **Speedup** = 56 / 13 = 4.3

- Number of images: 15.000
  - Execution time: 18 seconds
  - Processing speed: 840 images/second
  - **Speedup** = 79 / 18 = 4.38

- Using 10 processes:
  - Number of images: 2.000
    - Execution time: 2 seconds
    - Processing speed: 950 images/second
    - **Speedup** = 12 / 2 = 6

  - Number of images: 5.000
    - Execution time: 4 seconds
    - Processing speed: 1311 images/second
    - **Speedup** = 28 / 4 = 7

  - Number of images: 7.000
    - Execution time: 5 seconds
    - Processing speed: 1370 images/second
    - **Speedup** = 39 / 5 = 7.8

  - Number of images: 10.000
    - Execution time: 7 seconds
    - Processing speed: 1403 images/second
    - **Speedup** = 56 / 7 = 8

  - Number of images: 15.000
    - Execution time: 10 seconds
    - Processing speed: 1454 images/second
    - **Speedup** = 79 / 10 = 7.9

- Using 16 processes:
  - Number of images: 2.000
    - Execution time: 1 second
    - Processing speed: 1353 images/second
    - **Speedup** = 12 / 1 = 12

  - Number of images: 5.000
    - Execution time: 3 seconds
    - Processing speed: 1599 images/second
    - **Speedup** = 28 / 3 = 9.33

  - Number of images: 7.000
    - Execution time: 4 seconds
    - Processing speed: 1595 images/second
    - **Speedup** = 39 / 4 = 9.75

- Number of images: 10.000
  - Execution time: 6 seconds
  - Processing speed: 1722 images/second
  - **Speedup** = 56 / 6 = 9.33

- Number of images: 15.000
  - Execution time: 8 seconds
  - Processing speed: 1763 images/second
  - **Speedup** = 79 / 8 = 9.87

- Number of images: 200.000
  - Execution time: 113 seconds
  - Processing speed: 1763 images/second
  - **Speedup** = 728 / 113 = 6.4

** These values are approximate*

6) Performance Analysis

Comparing the two approaches, we observe that the parallel program offers a significant improvement in performance compared to the sequential version. The speedup achieved by the parallel program indicates how many times faster it runs compared to the sequential program. As the number of processes increases, the speedup generally increases, indicating better utilization of resources and faster execution. However, beyond a certain point, adding more processes may not result in significant speedup due to overhead and resource contention. It is crucial to find the optimal balance between the number of processes and the workload to maximize performance.

7) Conclusion

The parallel program demonstrates the effectiveness of utilizing multiple processes to process images concurrently. By leveraging parallel processing, we can achieve substantial speedup and improve overall throughput. However, optimizing performance requires careful consideration of factors such as the number of processes, task granularity, and hardware capabilities. Additionally, while parallelization offers performance benefits, it also introduces challenges such as managing concurrency and minimizing overhead. Overall, transitioning to a parallel approach can lead to significant performance gains, especially for computationally intensive tasks like image processing.

8) Recommendations

Based on this analysis, we recommend utilizing the parallel program for image processing tasks requiring fast and efficient execution. To achieve optimal performance, it is essential to experiment with different configurations, including the number of processes and workload distribution. Additionally, monitoring system resource usage and tuning parameters accordingly can help maximize performance and scalability. Furthermore, continuous testing and optimization are necessary to adapt the parallel program to changing workloads and hardware environments.