Signal Processing System Design Laboratory (EE69205)

LAB REPORT

by

YASH ANAND

(Roll No.- **23EE65R22**)

SIGNAL PROCESSING AND MACHINE LEARNING

Department of ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR
Kharagpur - 721302, India
September, 2023

|| Experiment 5 ||
|| Implementation of the moving window fast 2D median filter ||

AIM
To implement a fast 2D median filter using parallel programming in Python. This implementation aims to leverage parallel processing to improve the efficiency of the operation. Additionally, to estimate the space, time, and execution complexity of the implemented algorithm.

FUNCTION USED:

- **concurrent.futures.ThreadPoolExecutor():** This class creates a pool of worker threads that can be used to execute tasks in parallel.
- **skimage.io:** This library provides functions for reading and writing images
- **numpy.zeros_like():** This function creates a new array with the same shape and type as the given array, but with all elements initialized to zero.

OBSERVATION:

→ Each pixel's median value is computed concurrently, which can significantly speed up the filtering process.

→ the code uses a histogram to calculate the median value of the pixels in a tile. This is a more efficient approach than simply sorting the pixels and taking the middle value, especially for large tiles..

→ The code uses a single output image to store the filtered results. This is more efficient than using a separate output image for each tile, as it reduces the amount of memory that needs to be allocated.

→ The speedup of the parallel algorithm increases with the window size. This is because the parallel algorithm can distribute the workload more evenly for larger window sizes.

→ The code could be improved by using a larger number of threads and by filtering a larger set of images. This would provide more accurate and reliable results.

OUTPUT:

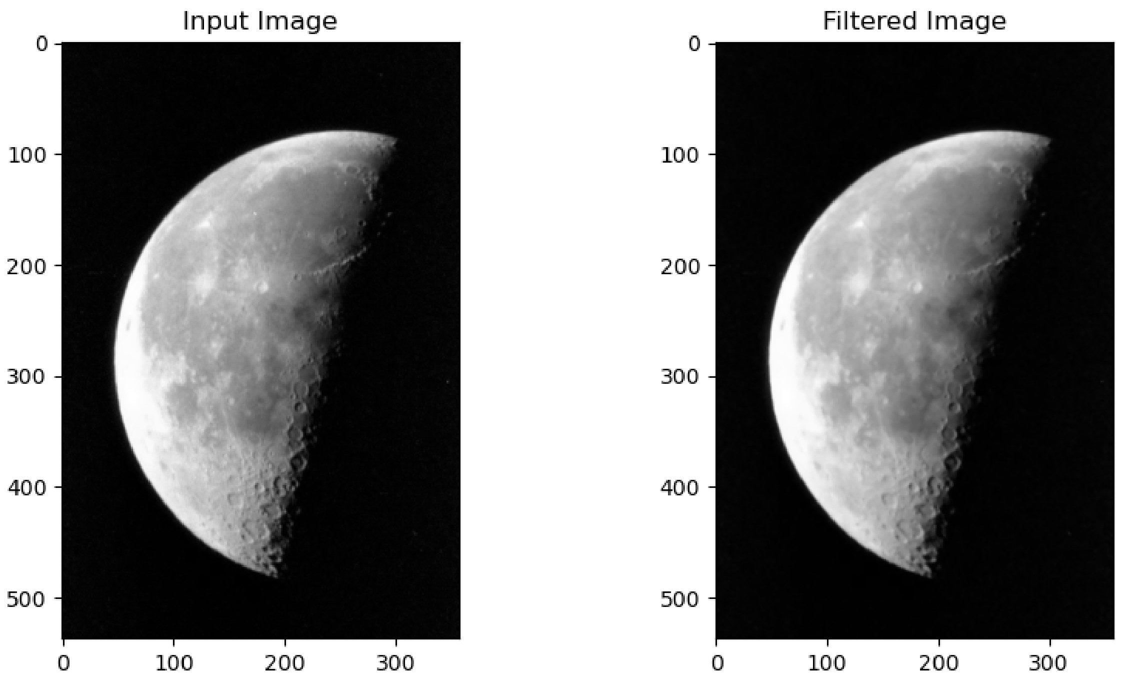

FIGURE 1: **C**omparative analysis for fast 2D median filter

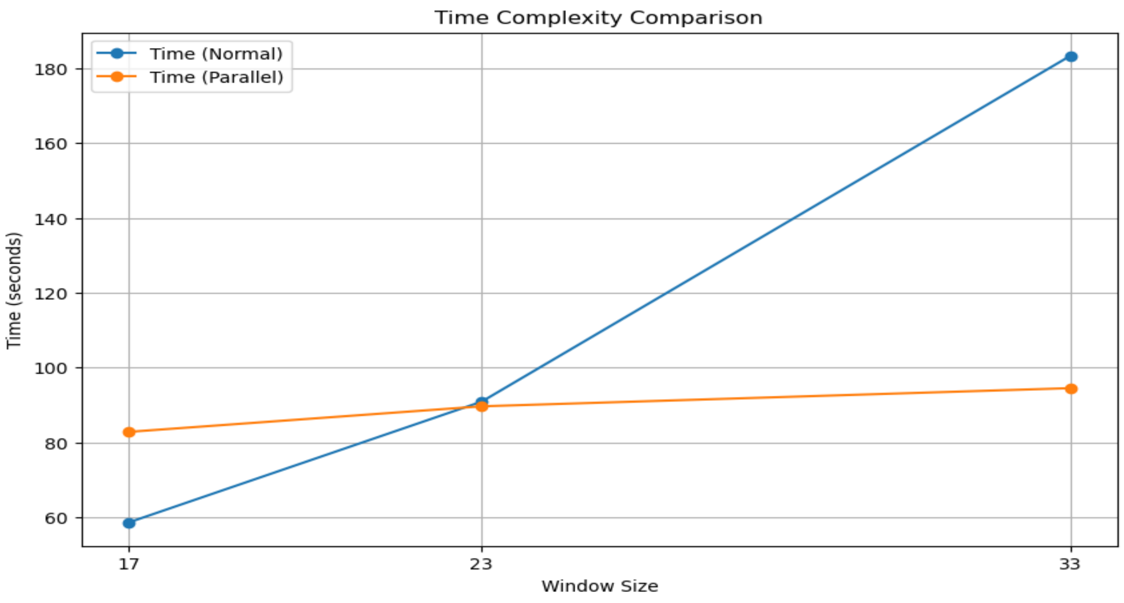

FIGURE 2: **T**ime Complexity Comparison

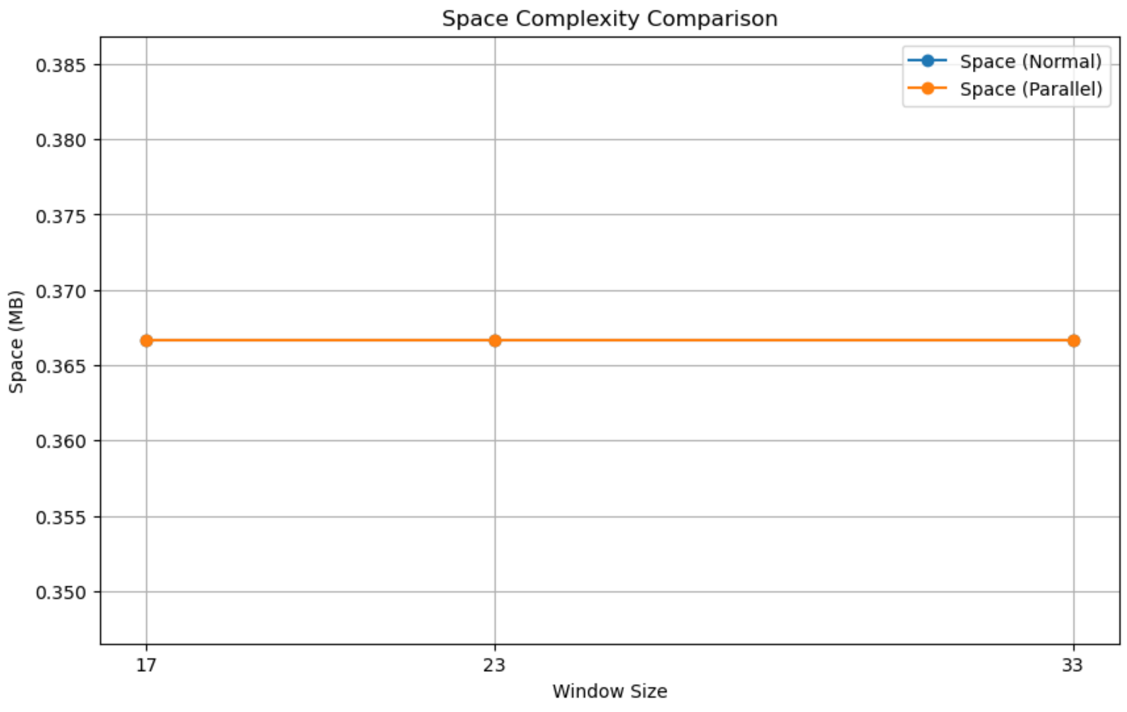

FIGURE 3: **S**pace Complexity Comparison