UNIVERSITY OF SCIENCE AND TECHNOLOGY OF HANOI

POSTGRADUATE SCHOOL

Advanced Programming for HPC
Project Report

Student: **Nguyễn Xuân Lộc**

ID: **M21.ICT.007**

Table of contents

I. Project explaination

For all implementations in CPU, GPU, and GPU with shared memory. I used the same structure for easy comparison and debug. However, the drawback of this structure is It may not be most efficient because it is required to read and write to memory multiple time.

The structure of my program is divided into 4 main functions which are

1. Calculate HSV image. In this case, we need only V value.
2. Using V values from HSV images, a function generate values of 4 windows for each pixel
3. A function calculates 4 Standard deviations of each pixel then output the window number
4. Kuwahara filter calculate the values for each pixel based on the window
5. An additional function to add padding to HSV and RGB images

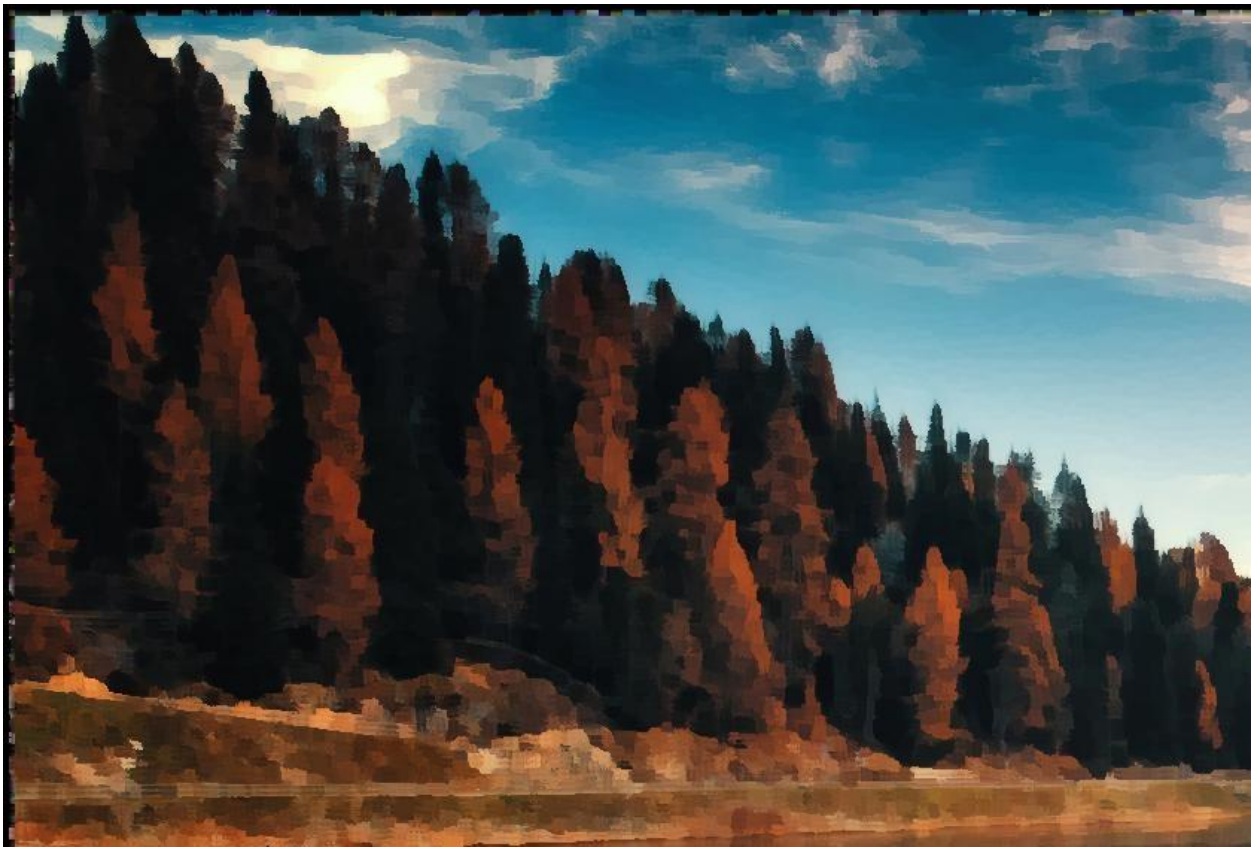
Please note that:

- For CPU implementation, the program needs to iterate to each pixel to calculate the value then generate an output
- For GPU implementation, the program processes each pixel in parallel however still using main GPU memory.
- For GPU with shared memory implementation, each function loads its input into shared memory, then calculate the output from there.

II. Result

Original Image

Result of Kuwahara filtering by CPU, GPU, and GPU with shared mem (output is same)



The result when zoomed in

III. Measure speed up

1. Speed comparision between CPU, GPU, and GPU with shared memory

#	**Implementation**	**Time to run**
1	CPU	548 second
2	GPU	2.39 second
3	GPU with shared memory	3.11 second

The GPU implementation is significant faster than CPU implementation. However, it is surprised that the GPU with shared memory is slower.

2. Explaintion of why GPU with shared memory speed is slower

As mentioned in the first part, the structure of the program is divided into 4 main functions for easier to debug and test. However, this structure requires read and write to memory when the function start and end.

The current shared memory implementation only utilized the shared memory within the function. That is the function load the input data into shared memory, then the calculation using the shared memory, then the output is written back to the main memory. This create an additional step which is copy from main memory to shared memory in comparison to GPU implementation. However, it is not utilized the transfer of IO between functions using shared memory. This is a possible improvement.

IV. Possible Improvement for shared memory implementation

For the GPU with shared memory, the possible method to improve the performance is to utilize the shared memory in IO between functions. To do it, I plan to write a kernel to handle all the shared memory, then call the function to calculate the output. The output then will be written to shared memory. Only the final output of the transformed image will be written back to the main memory of the GPU.