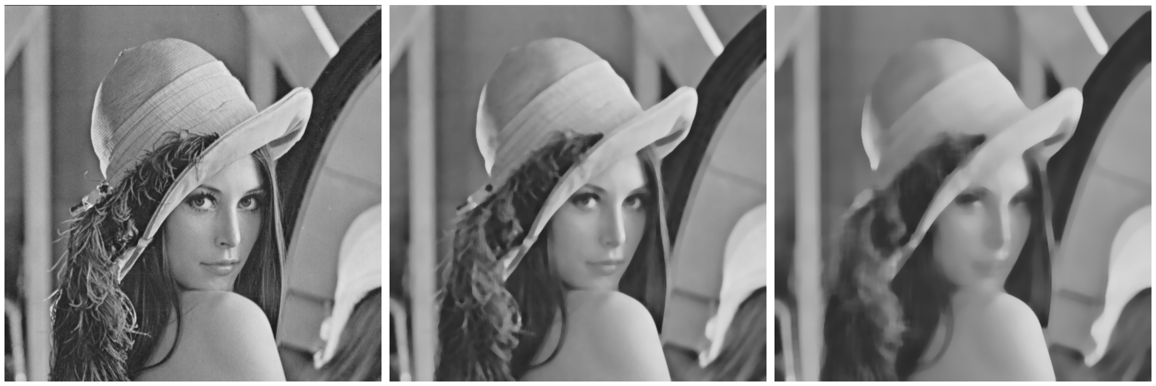
CS 7080 High Performance Computing
Homework 5
Intro to CUDA - Median Filter

Haoxiang Zhang
PowPrint: hzny2

Due: December 9, 2021

1 Homework Description

Homework 5, this is the warm up project for getting familiar with CUDA programming. This homework, we are going to implement median filter based CUDA.In the Fig.1, it shows how the Median Filter works.



(a) Input Lena.pgm image (b) Output image - WindowSize: 7 (c) Output image - WindowSize: 15

Figure 1: Median Filter Algorithm Example

2 Homework Details

For CUDA programming, we have to understand the details about CUDA. CUDA programming has HOST and DEVICE which host is the CPU and its memory and which device is the GPU and its memory. The basic processing flow is:

- Copy input data from CPU memory to GPU memory
- Load GPU program and execute caching data on chip for performance
- Copy results from GPU memory to CPU memory

GPU programming is kind of like a multi-threading programming. GPU has many SM(Stream Multiprocessors). There is a CUDA grid. This grid has multiple CUDA Blocks - multiprocessing units. Those blocks are broken up into CUDA threads. The information below is about how to get the value of the the blocks and threads:

- blockDim.x, blockDim.y, blockDim.z - To get a block's size
- blockIdx.x, blockIdx.y, blockIdx.z - To get a block's position
- threadIdx.x, threadIdx.y, threadIdx.z - To get a thread's position in a block

Therefore, we can easily to find the value at any of the position. The code below shows how to find the the value of the position.

```
// position in a block (local position)
const int xx = threadIdx.x;
const int yy = threadIdx.y;
const int zz = threadIdx.z;

// position in grid (global position)
const int x = blockDim.x * blockIdx.x + xx;
const int y = blockDim.y * blockIdx.y + yy;
const int z = blockDim.z * blockIdx.z + zz;

// position in 2 dimension space
const int 2d_pos = y * blockDim.x + x;

// position in 3 dimension space
const int 3d_pos = z * (blockDim.y * blockDim.x) + y * blockDim.x + x;
```

For the kernel function, if there is a **__global__** the beginning of the function, this means this function will be run in GPU. In the example code below, we can found that there is a <<<**grid, block**>>> when call the kernel function. The grid is telling the GPU for how many blocks you need; and block is telling the GPU for how many threads in each block.

```
// kernel function
__global__ void median_filter_gpu(unsigned char* input_img, unsigned char* output_img, int img_w, int img_h, int wind_size)

// to call the kernel function
const dim3 grid  (grid_divide(width, BLOCK_WIDTH), grid_divide(height, BLOCK_HEIGHT), 1);
const dim3 block (BLOCK_WIDTH, BLOCK_HEIGHT, 1);
median_filter_gpu<<<grid, block>>>(d_img, d_out, width, height, wind_size);
```

How to define the size of the grid and the block

For the given image - **lena.pgm**, the image size is $512 \times 512 \times 1$ - a gray-scaled image. Usually set the **BlockWidth** and **BlockHeight** as 16. Then, `[image_width / BlockWidth, image_height / BlockHeight, 1]` will be the grid size. And `[BlockWidth, BlockHeight, 1]` will be the block size. This could make sure that each thread in blocks could deal with each pixel's median value.

Examples image of the image and output images

In the Fig.2, it shows the original input image. And Fig.4 is showing the output with different window size. We can find that the image is getting more blur by increase the window size. This filter method is usually used on noise removing.

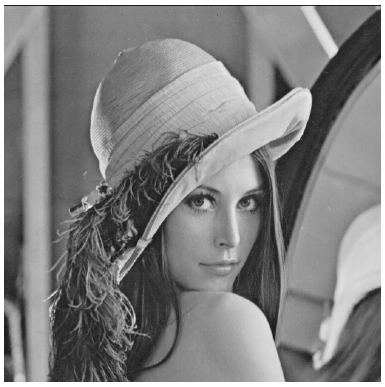
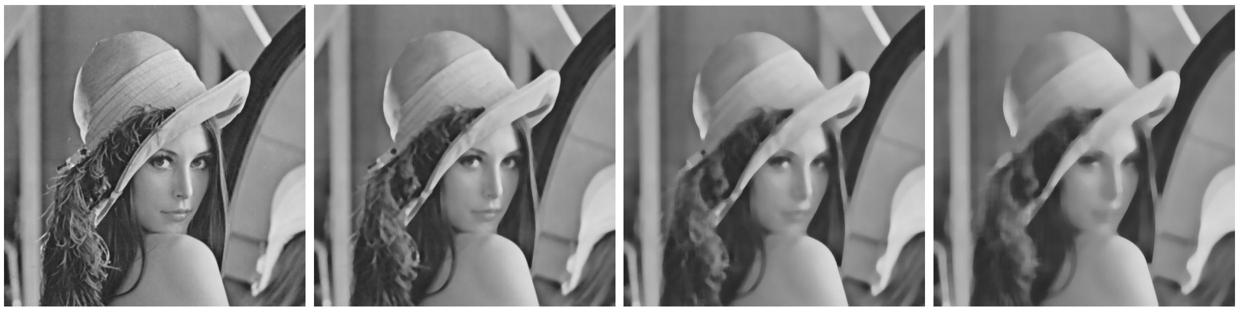


Figure 2: Input Image



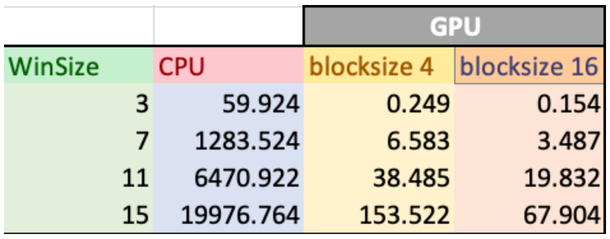
(a) Window Size: 3×3 (b) Window Size: 7×7 (c) Window Size: 11×11 (d) Window Size: 15×15

Figure 3: Output with Different Window Size

Timing Trends

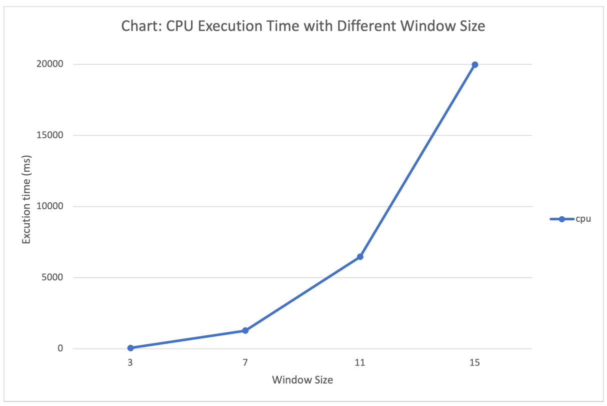
In the Fig.4a, we can found that CPU execution time increase crazily. This is because each pixel need to find larger and larger amount of the neighbors and sort them, which cause more time to finish the work. However, using CUDA, each threads in blocks will get the median value in parallel which saves plenty of time.

For different Block Size, based on the time in Fig.4c, we can found that the small block size cause more time. The reason I guess is because each block would do synchronization after all of the threads finish, which means more block number cause more time to synchronization. That's why smaller block size spend more time.

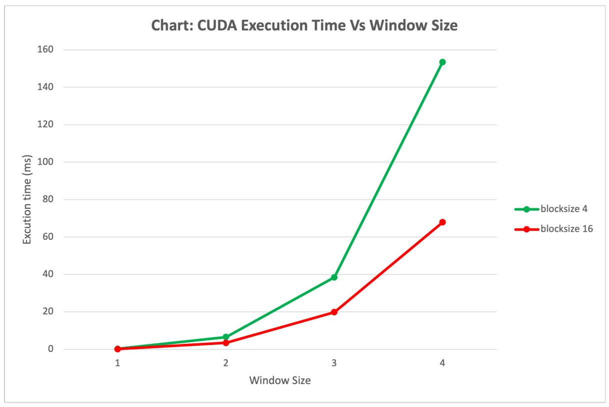


		GPU	
WinSize	CPU	blocksize 4	blocksize 16
3	59.924	0.249	0.154
7	1283.524	6.583	3.487
11	6470.922	38.485	19.832
15	19976.764	153.522	67.904

(a) Table for Timing Trends



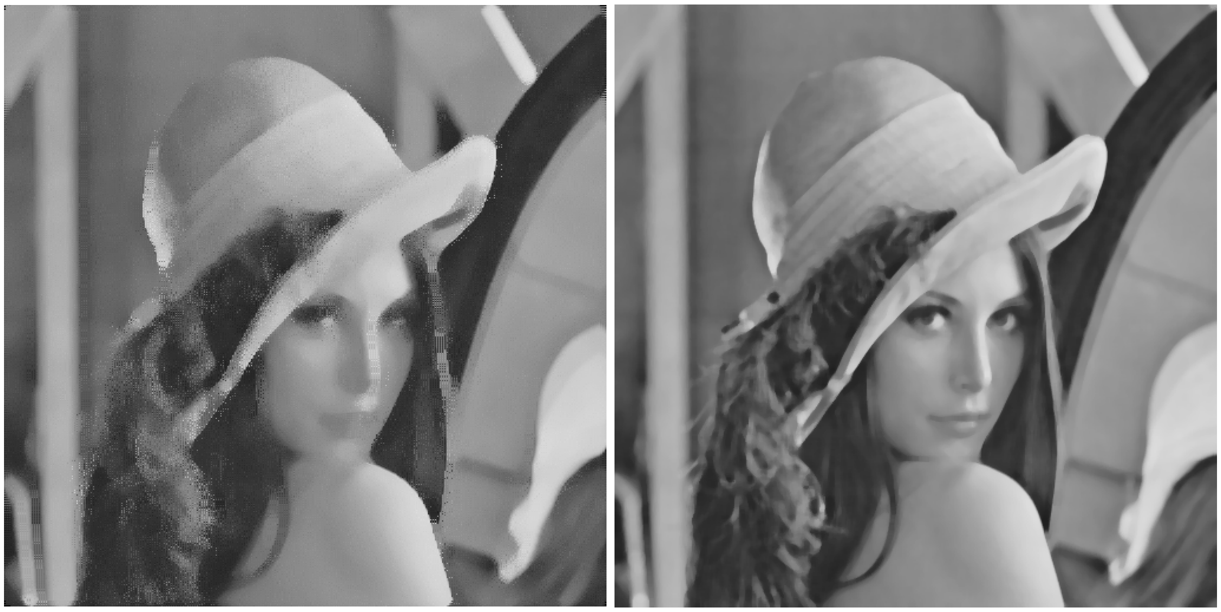
(b) CPU



(c) CUDA

Figure 4: Chart for Timing Trends With Different Cases

3 Problems & Solutions



(a) Wrong Output (b) Correct Output

Figure 5: Problem

I had a wrong result like Fig.5a. It is easily to found there is a lot of noise points in the image. Median Filter is used to remove noise, but the output create more noise, which means there is something wrong about the code.

Then, after I search the problem online, and I found after I change the image data type from `unsigned short* []` to `unsigned char* []`, the result show correctly. The reason I guess is that the size of a short is 16 bits and a char is 8 bits, so that's probably why.