F21_GPU_Proj01 Report

Jiale Shi

I take the following public resources as my reference to do this job.
Without the help of these public resources, I could not finish this homework.

1. https://stackoverflow.com/questions/39336574/box-filter-with-cuda-c
2. https://www.nvidia.com/content/nvision2008/tech_presentations/Game_Developer_Track/NVISION08-Image_Processing_and_Video_with_CUDA.pdf

The project is a box linear filter for image processing.
https://en.wikipedia.org/wiki/Box_blur

Sunflower_3.jpg is the output after box linear filter.

CUDA kernel launch with (174,122) blocks of (16,16) threads in my code.

I find that the matr_multiply.cu cannot include pointer_2d_matrix.h.
Therefore, I cannot call Matrix in matr_multiply.cu.
In order to solve the problem, I redefine arrays in the main.cpp and pass array to the cuda_kernel_wrapper().

```
float  *data_Red, *data_Blue, *data_Green;
data_Red = (float *)malloc(Width*Height*sizeof(float));
data_Blue = (float *)malloc(Width*Height*sizeof(float));
data_Green = (float *)malloc(Width*Height*sizeof(float));
```

I pass the array individually to cuda_kernel_wrapper() to do box linear filter.

```
// this eventually calls cuda kernel
cout << "Process Red:" << endl;
cuda_kernel_wrapper(data_Red, Width, Height);
```

```
cout << "Process Blue:" << endl;
cuda_kernel_wrapper(data_Blue, Width, Height);
cout << "Process Green:" << endl;
cuda_kernel_wrapper(data_Green, Width, Height);
```

cuda_kernel_wrapper() uses the d_filter kernel.

__global__ void d_filter(float *g_idata, float *g_odata, unsigned int width, unsigned int height)

Inside the d_filter, we use shared memory.

Performance Test:
I cut the original Sunflower.jpg to create smaller images.
Sunflower.jpg
Image width: 2434 pixels
Image height: 1697 pixels
Process Red:
CUDA kernel launch with 174,122 blocks of 16,16 threads
Time to run the kernel: **0.325792 ms**

Process Blue:
CUDA kernel launch with 174,122 blocks of 16,16 threads
Time to run the kernel: **0.276480 ms**

Process Green:
CUDA kernel launch with 174,122 blocks of 16,16 threads
Time to run the kernel: **0.281760 ms**

Sunflower_M.jpg
Image width: 1930 pixels
Image height: 1396 pixels

Process Red:
CUDA kernel launch with 138,100 blocks of 16,16 threads
Time to run the kernel: **0.231648 m**s

Process Blue:
CUDA kernel launch with 138,100 blocks of 16,16 threads
Time to run the kernel: **0.185824 ms**

Process Green:
CUDA kernel launch with 138,100 blocks of 16,16 threads
Time to run the kernel: **0.184064 ms**

Sunflower_S.jpg
Image width: 1537 pixels
Image height: 1063 pixels

Process Red:
CUDA kernel launch with 110,76 blocks of 16,16 threads
Time to run the kernel: **0.156512 ms**

Process Blue:
CUDA kernel launch with 110,76 blocks of 16,16 threads
Time to run the kernel: **0.113984 ms**

Process Green:
CUDA kernel launch with 110,76 blocks of 16,16 threads
Time to run the kernel: **0.106848 ms**

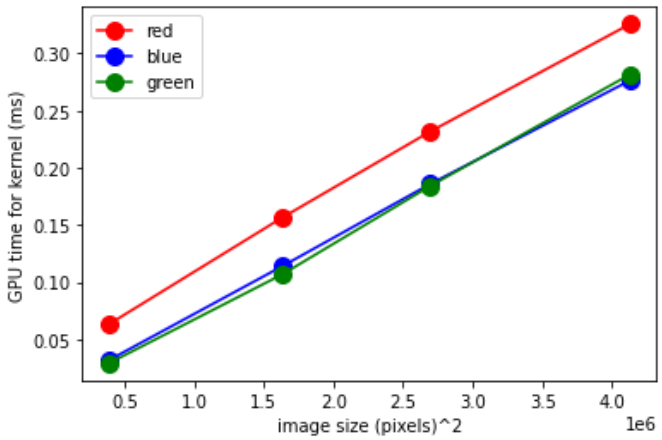
Sunflower_XS.jpg
Image width: 644 pixels
Image height: 601 pixels

Process Red:
CUDA kernel launch with 47,43 blocks of 16,16 threads
Time to run the kernel: **0.063040 ms**

Process Blue:
CUDA kernel launch with 47,43 blocks of 16,16 threads
Time to run the kernel: **0.031680 ms**

Process Green:
CUDA kernel launch with 47,43 blocks of 16,16 threads
Time to run the kernel: **0.028992 ms**

Conclusion:
Smaller size images need smaller blocksPerGrid (fewer blocks) and the time to run the kernel would be shorter.


red
blue
green
GPU time for kernel (ms)
0.30
0.25
0.20
0.15
0.10
0.05
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
image size (pixels)^2
1e6