# Image Convolution Grade

## Grade Summary (History) (/grade/history/5986)

| | |
|---|---|
| Created: | less than a minute ago (2022-04-04 02:11:18 +0000 UTC) |
| Total Score: | 100 out of 100 points |
| Coding Score: | 100 out of 100 points |
| Questions Score: | 0 |

## Program Code

```
#include    <wb.h>

#define wbCheck(stmt)                                     \
    do {                                                  \
        cudaError_t err = stmt;                           \
        if (err != cudaSuccess) {                         \
            wbLog(ERROR, "Failed to run stmt ", #stmt);     \
            return -1;                                    \
        }                                                 \
    } while(0)

#define Mask_width  5
#define Mask_radius Mask_width / 2
#define TILE_WIDTH  16
#define SIZE        (TILE_WIDTH + Mask_width - 1)

//@@ INSERT CODE HERE
__global__
void Image_Convolution (float * I, const float * __restrict__ M, float *
{
    __shared__ float N_ds[SIZE][SIZE];
```

```
22
23      int block_x = blockIdx.x,  block_y = blockIdx.y;
24      int thread_x = threadIdx.x, thread_y = threadIdx.y;
25
26      //lookup variable designations and formula uses + borrow loops from g
27      for (int k = 0; k < channels; ++k) {
28          int dest  = thread_y * TILE_WIDTH + thread_x;
29          int destX = dest % SIZE;
30          int destY = dest / SIZE;
31          int srcY  = block_y * TILE_WIDTH + destY - Mask_radius;
32          int srcX  = block_x * TILE_WIDTH + destX - Mask_radius;
33          int src   = (srcY * width + srcX) * channels + k;
34
35          if (srcY >= 0 && srcY < height && srcX >= 0 && srcX < width)
36              N_ds[destY][destX] = I[src];
37          else
38              N_ds[destY][destX] = 0.0;
39
40          dest  = thread_y * TILE_WIDTH + thread_x + TILE_WIDTH * TILE_WIDT
41          destY = dest / SIZE;
42          destX = dest % SIZE;
43          srcY  = block_y * TILE_WIDTH + destY - Mask_radius;
44          srcX  = block_x * TILE_WIDTH + destX - Mask_radius;
45          src   = (srcY * width + srcX) * channels + k;
46
47          if (destY < SIZE) {
48              if (srcY >= 0 && srcY < height && srcX >= 0 && srcX < width)
49                  N_ds[destY][destX] = I[src];
50              else
51                  N_ds[destY][destX] = 0.0;
52          }
53          __syncthreads();
54
55          float accum = 0;
56          for (int y = 0; y < Mask_width; ++y)
57              for (int x = 0; x < Mask_width; ++x)
58                  accum += N_ds[thread_y + y][thread_x + x] * M[y * Mask_wi
59
60          int x = block_x * TILE_WIDTH + thread_x;
61          int y = block_y * TILE_WIDTH + thread_y;
62
63          if (y < height && x < width)
64              P[(y * width + x) * channels + k] = min(max(accum, 0.0), 1.0)
65
66          __syncthreads();
```

```
67      }
68  }
69
70  int main (int argc, char * argv[ ])
71  {
72      wbArg_t arg;
73      int maskRows;
74      int maskColumns;
75      int imageChannels;
76      int imageWidth;
77      int imageHeight;
78      char * inputImageFile;
79      char * inputMaskFile;
80      wbImage_t inputImage;
81      wbImage_t outputImage;
82      float * hostInputImageData;
83      float * hostOutputImageData;
84      float * hostMaskData;
85      float * deviceInputImageData;
86      float * deviceOutputImageData;
87      float * deviceMaskData;
88
89      arg = wbArg_read(argc, argv); /* parse the input arguments */
90
91      inputImageFile = wbArg_getInputFile(arg, 0);
92      inputMaskFile = wbArg_getInputFile(arg, 1);
93
94      inputImage = wbImport(inputImageFile);
95      hostMaskData = (float *) wbImport(inputMaskFile, &maskRows, &maskColu
96
97      assert(maskRows == 5); /* mask height is fixed to 5 in this mp */
98      assert(maskColumns == 5); /* mask width is fixed to 5 in this mp */
99
100     imageWidth = wbImage_getWidth(inputImage);
101     imageHeight = wbImage_getHeight(inputImage);
102     imageChannels = wbImage_getChannels(inputImage);
103
104     outputImage = wbImage_new(imageWidth, imageHeight, imageChannels);
105
106     hostInputImageData = wbImage_getData(inputImage);
107     hostOutputImageData = wbImage_getData(outputImage);
108
109     wbTime_start(GPU, "Doing GPU Computation (memory + compute)");
110
111     wbTime_start(GPU, "Doing GPU memory allocation");
```

```
    cudaMalloc((void **) &deviceInputImageData, imageWidth * imageHeight
    cudaMalloc((void **) &deviceOutputImageData, imageWidth * imageHeight
    cudaMalloc((void **) &deviceMaskData, maskRows * maskColumns * sizeof
    wbTime_stop(GPU, "Doing GPU memory allocation");

    wbTime_start(Copy, "Copying data to the GPU");
    cudaMemcpy(deviceInputImageData, hostInputImageData, imageWidth * ima
    cudaMemcpy(deviceMaskData, hostMaskData, maskRows * maskColumns * siz
    wbTime_stop(Copy, "Copying data to the GPU");

    wbTime_start(Compute, "Doing the computation on the GPU");
    //@@ INSERT CODE HERE
    dim3 dimGrid(ceil((float) imageWidth / TILE_WIDTH), ceil((float) imag
    dim3 dimBlock(TILE_WIDTH, TILE_WIDTH, 1);
    Image_Convolution<<<dimGrid, dimBlock>>>(deviceInputImageData, device

    wbTime_stop(Compute, "Doing the computation on the GPU");

    wbTime_start(Copy, "Copying data from the GPU");
    cudaMemcpy(hostOutputImageData, deviceOutputImageData, imageWidth * i
    wbTime_stop(Copy, "Copying data from the GPU");

    wbTime_stop(GPU, "Doing GPU Computation (memory + compute)");

    wbSolution(arg, outputImage);

    cudaFree(deviceInputImageData);
    cudaFree(deviceOutputImageData);
    cudaFree(deviceMaskData);

    free(hostMaskData);
    wbImage_delete(outputImage);
    wbImage_delete(inputImage);

    return 0;
}
```