(/mp/5986?show=code) Image Convolution Attempt

Attempt Summary

Submit Attempt for Grading

Remember to answer the questions before clicking.

Dataset Id:	5
Created:	less than a minute ago (2022-04-04 02:10:51 +0000 UTC)
Status:	Correct solution for this dataset.

Timer Output

Kind	Location	Time (ms)	Message
GPU	main.cu::109	6.475455	Doing GPU Computation (memory + compute)
GPU	main.cu::111	0.992476	Doing GPU memory allocation
Copy	main.cu::117	1.877611	Copying data to the GPU
Compute	main.cu::122	0.810309	Doing the computation on the GPU
Copy	main.cu::130	2.766035	Copying data from the GPU

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];

    int block_x = blockIdx.x,  block_y = blockIdx.y;
    int thread_x = threadIdx.x, thread_y = threadIdx.y;

    //lookup variable designations and formula uses + borrow loops from g
    for (int k = 0; k < channels; ++k) {
        int dest  = thread_y * TILE_WIDTH + thread_x;
        int destX = dest % SIZE;
        int destY = dest / SIZE;
        int srcY  = block_y * TILE_WIDTH + destY - Mask_radius;
        int srcX  = block_x * TILE_WIDTH + destX - Mask_radius;
        int src   = (srcY * width + srcX) * channels + k;

        if (srcY >= 0 && srcY < height && srcX >= 0 && srcX < width)
            N_ds[destY][destX] = I[src];
        else
            N_ds[destY][destX] = 0.0;

        dest  = thread_y * TILE_WIDTH + thread_x + TILE_WIDTH * TILE_WIDT
        destY = dest / SIZE;
        destX = dest % SIZE;
        srcY  = block_y * TILE_WIDTH + destY - Mask_radius;
        srcX  = block_x * TILE_WIDTH + destX - Mask_radius;
        src   = (srcY * width + srcX) * channels + k;

        if (destY < SIZE) {
            if (srcY >= 0 && srcY < height && srcX >= 0 && srcX < width)
                N_ds[destY][destX] = I[src];
            else
                N_ds[destY][destX] = 0.0;
        }
        __syncthreads();

        float accum = 0;
        for (int y = 0; y < Mask_width; ++y)
            for (int x = 0; x < Mask_width; ++x)
                accum += N_ds[thread_y + y][thread_x + x] * M[y * Mask_wi

        int x = block_x * TILE_WIDTH + thread_x;
        int y = block_y * TILE_WIDTH + thread_y;

        if (y < height && x < width)
            P[(y * width + x) * channels + k] = min(max(accum, 0.0), 1.0)

        __syncthreads();
    }
}

int main (int argc, char * argv[ ])
{
    wbArg_t arg;
    int maskRows;
    int maskColumns;
    int imageChannels;
    int imageWidth;
    int imageHeight;
    char * inputImageFile;
    char * inputMaskFile;
    wbImage_t inputImage;
    wbImage_t outputImage;
    float * hostInputImageData;
    float * hostOutputImageData;
    float * hostMaskData;
    float * deviceInputImageData;
    float * deviceOutputImageData;
    float * deviceMaskData;

    arg = wbArg_read(argc, argv); /* parse the input arguments */

    inputImageFile = wbArg_getInputFile(arg, 0);
```

```
  inputMaskFile = wbArg_getInputFile(arg, 1);

  inputImage = wbImport(inputImageFile);
  hostMaskData = (float *) wbImport(inputMaskFile, &maskRows, &maskColu

  assert(maskRows == 5); /* mask height is fixed to 5 in this mp */
  assert(maskColumns == 5); /* mask width is fixed to 5 in this mp */

  imageWidth = wbImage_getWidth(inputImage);
  imageHeight = wbImage_getHeight(inputImage);
  imageChannels = wbImage_getChannels(inputImage);

  outputImage = wbImage_new(imageWidth, imageHeight, imageChannels);

  hostInputImageData = wbImage_getData(inputImage);
  hostOutputImageData = wbImage_getData(outputImage);

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

  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;
}
```

Designed and architected by Abdul Dakkak (https://www.dakkak.dev/).