Advaned Programming fo HPC - Report labwork 4

NGUYEN TAT HUNG

November 4, 2021

Implementation

```
__global__ void grayscale2D(uchar3 *input, uchar3 *output, int width, int height) {
        int x = threadIdx.x + blockIdx.x + blockDim.x;
        int y = threadIdx.y + blockIdx.y + blockDim.y;
        if (x > width || y > height){
            printf("x, y are outside the width, height");
            return;
        }
        int tid = x + y * width;
        output[tid].x = (input[tid].x + input[tid].y + input[tid].z) / 3;
        output[tid].z = output[tid].y = output[tid].x;
}
void Labwork::labwork4_GPU() {
    // Calculate number of pixels
    int pixelCount = inputImage->width * inputImage->height;
    //char *hostInput = inputImage->buffer; // Perfect version
    char *hostInput = (char*) malloc(inputImage->width * inputImage->height * 3); // Test
    char *hostOutput = new char[inputImage->width * inputImage->height * 3]; // Test versio
    outputImage = static_cast<char *>(malloc(pixelCount * 3));
    for (int j = 0; j < 100; j++) {     // let's do it 100 times, otherwise it$
        # pragma omp parallel for
        for (int i = 0; i < pixelCount; i++) {
            outputImage[i * 3] = (char) (((int) inputImage->buffer[i * 3] + (int) inputIma
            outputImage[i * 3 + 1] = outputImage[i * 3];
            outputImage[i * 3 + 2] = outputImage[i * 3];
        }
    }

    // Allocate CUDA memory
    uchar3 *devInput;
    uchar3 *devOutput;
    //cudaMalloc(&devInput, pixelCount*3); // Perfect version
    cudaMalloc(&devInput, pixelCount * sizeof(uchar3)); // Test version
    //cudaMalloc(&devOutput, pixelCount*3); // Perfect version
    cudaMalloc(&devOutput, pixelCount * sizeof(float)); // Test version

    // Copy CUDA Memory from CPU to GPU
    //cudaMemcpy(devInput, hostInput, pixelCount*3, cudaMemcpyHostToDevice); // Perfect ve
    cudaMemcpy(devInput, hostInput, pixelCount * sizeof(uchar3), cudaMemcpyHostToDevice);


    // Processing
```

```
    dim3 blockSize = dim3(32, 32);
    dim3 gridSize = ((int) ((inputImage->width + blockSize.x - 1)/blockSize.x), (int)((inpu
    grayscale2D<<<gridSize, blockSize>>>(devInput, devOutput, inputImage->width, inputImage

    // Copy CUDA Memory from GPU to CPU
    //cudaMemcpy(outputImage, devOutput, pixelCount*3, cudaMemcpyDeviceToHost); // Perfect
    cudaMemcpy(hostOutput, devOutput, pixelCount*sizeof(float), cudaMemcpyDeviceToHost); /

    // Cleaning
    //free(hostInput);
    cudaFree(devInput);
    cudaFree(devOutput);
}
```

Result

Figure 1: Original input image

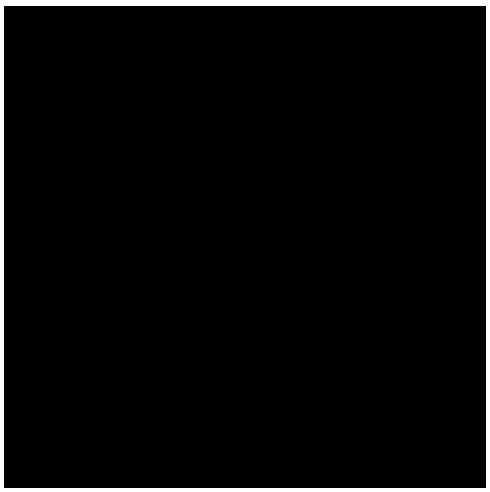

Figure 2: Output image