Filip Zachrisson
970820-9110

Fredrik Lind
960325-4195

Lab 3 - Image Blurring

DV2581 - Multicore Programming

Acknowledgements

- Terms that we will be using that refers to our program:
    - BLOCK_SIZEX/Y - Number of threads per block in dim x/y.
- All output was disabled when gathering results.
- The testing was done on a 4000x3000 pixel picture.
- All testing was done on the school computers (GTX 1080).
- Used command in Windows PowerShell:
    - *Measure-Command {start-process '.\blurpar.exe' -Wait}*

Introduction

This report will explain how we went about implementing Image Blurring on the GPU by using parallel programming with CUDA.

Method

First we tried to "manually" load the data from the jpg file, but that was tiresome and a bit more tricky than assumed. So we went with the given library instead. But the subtask wanted us to transform an Array of Structured (AoS) rgb-values to a Structure of Arrays (SoA), which means that we need to get the rgb-values to form three separate arrays of red, green and blue. However, the given library was already organizing the colors in this format, so we had to first copy the rgb-values while transforming them into an AoS-format, then sending this structured array to the GPU and letting it transform it back to the SoA-format. Note that this transformation by the kernel call is not thread divergent because all the threads in all warps run the same if-statement in just one call.

After that we created another call to the GPU which will be called one time for each array (color-value) where a 5x5 mask was sent with the arrays. With this mask we were able to calculate the blur on each color and then return the blurred array.

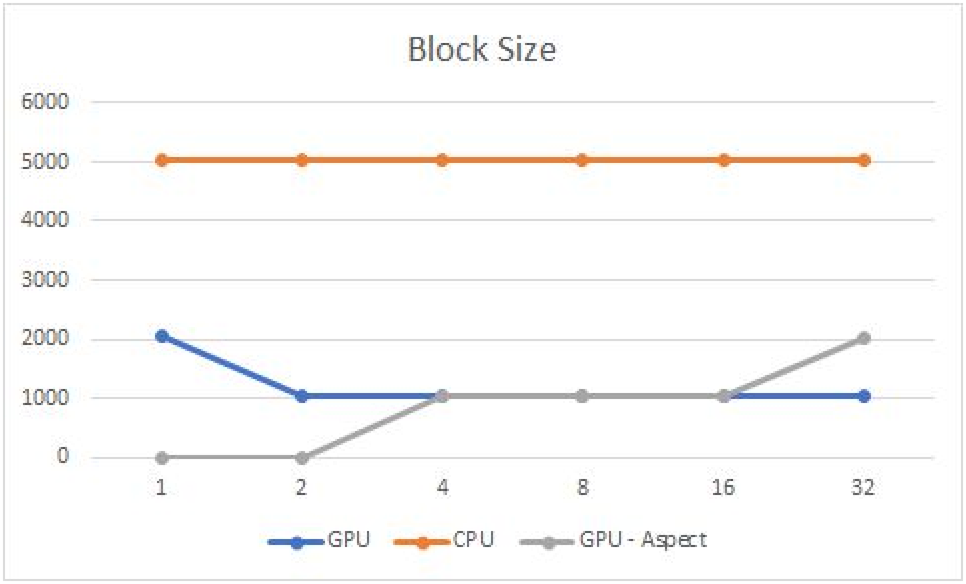
We did not include any thread coarsening because we want the program to run when the number of threads are equal to the number of pixels. This means that thread-coarsening only helps with pictures that are bigger than 65535x65535 and we feel that that size is beyond the "normal" use.

Findings

The following figure is done on a 4000x3000 image as mentioned in the acknowledgements. However, we had in plan to try out the code on a bigger imagine because the program ran so fast that it was hard to get good data. This is because it takes Windows at least around 1000ms to load and prepare the .exe file, so it was not possible to get data below 1000ms. But we were not sure if it was necessary to try bigger images because the task wants us to run the specific 4000x3000 image, so we skipped it.

The maximum block size is 32 because 32 * 32 (dim3(32, 32)) reaches the maximum *threads per block* limit.

Figure 1. A time comparison between the CPU and GPU, when increasing the block size from 1 to 32 or matching the block size to the aspect ratio of the image.



We gain a tremendous speed up when using the GPU and the GPU-speed will differ depending on a matching aspect ratio or the size of the blocks.

Result

The data from *Figure 1* shows that you will gain a speedup while using the GPU instead of the CPU when blurring. However, the GPU will run most effectively as long as we have block sizes between 4 and 16 with the matching aspect ratio between the BLOCK_SIZEX/Y and the image size. This is because a matching aspect ratio will lead to no unused threads within a block, which means good padding. Note that the figure may be too small to analyze, therefore it is hard to see any difference between the block size from 2 to 16.