Last update: 31/10/2022

High Performance Systems Programming (ECE 415)
Department of Electrical and Computer Engineering
University of Thessaly

Teacher: Christos D. Antonopoulos

3rd Laboratory Exercise

Target : Getting familiar with the CUDA compilation environment, writing a simple CUDA program, experimenting with block and grid geometries, thread count problems, accuracy problems.

Background :
Convolution is used in many engineering and mathematical applications. For example, many image transform filters are essentially implemented as convolutions. In the right part of the image below you see the result of applying a Gaussian blur filter.

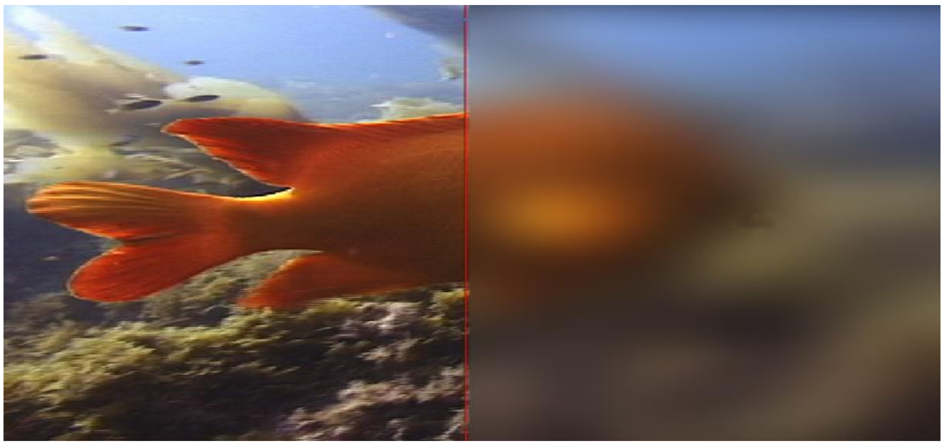

Figure 1: Applying gaussian blur to an image (http://www.borisfx.com/images/bcc3/gaussian_blur.jpg)

In mathematical terms convolution quantifies the result of "overlapping" two functions. We can think of it as a "mixing" operation that completes the result of point-by-point multiplication of one data set by another.

$$(\ ) = (\ *\ )(\ ) = \int\ (\ -\ ) *\ (\ )$$

In the discrete world, it can be written as:

$$(\ ) = (\ *\ )(\ ) = \sum\ (\ -\ )(\ )$$

Convolution can also be extended to 2 dimensions:

$$(\ ) = (\ *\ )(\ ,\ ) = \sum\sum\ (\ -\ ,\ \ -\ )(\ ,\ )$$

In the field of image processing, a convolutional filter is simply the point-to-point product of the filter weights by the pixels of the input image, within a window that encloses each output pixel. The figure below illustrates the process of applying a 3x3 mask to an input image to produce a pixel of the output image. In the next step, the mask will be moved over the input image to calculate a new pixel of the output image, and so on.

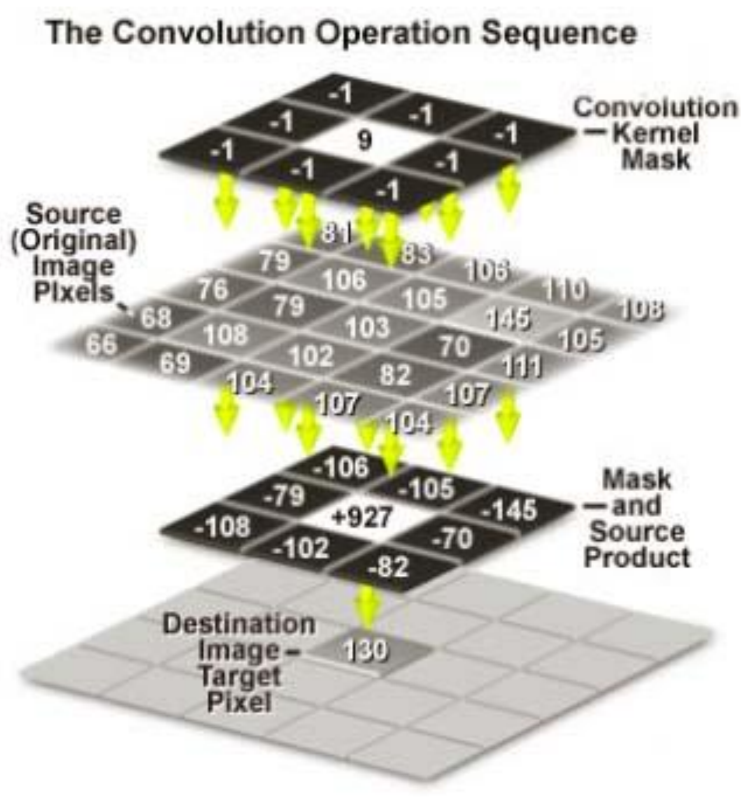
The Convolution Operation Sequence
Convolution — Kernel Mask
Source (Original) Image Pixels
Mask — and Source Product
+927
Destination Image - Target Pixel
130

Figure 1: Applying 2D convolution with a 3x3 mask to an image to produce a pixel of the image result.

(http://www.biomachina.org/courses/structures/01.html)

Typically, applying a 2D convolution filter to compute an output (pixel) value requires n*m multiplications, where n and m are the dimensions of the filter. Separable are two-dimensional filters that can be expressed as the composition of two one-dimensional filters, one of which is applied to the lines of the image and one to the columns. For example, applying the Sobel filter below
$\begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix}$ is equivalent to applying $\begin{bmatrix}1\\ 2\\ 1\end{bmatrix}$ followed by $[-1\ 0\ 1]$.
In this exercise you are asked to write the code that will implement a separable, two-dimensional filter over a two-dimensional array (which, let's say, corresponds to an image).

Steps :
0) Run deviceQuery and log the results. This step should be repeated for all exercises.

1) Look closely at the nvcc compiler run parameters by typing nvcc --help in one

terminal. Always make sure to compile code for the CPU with the highest degree of optimizations (-O4).

2) Write code that:

  a) Performs 2D convolution on an NxN image with float elements, using a filter containing random values, on the GPU. The result should also be stored in an NxN image. The size N is given by the command line at program execution, while the input image is initialized with random values. How the 2D convolution is done is given as code that runs on the CPU and by analyzing it you can understand its operation and transfer it to the GPU. FILTER_RADIUS should also be given from the command line while FILTER_LENGTH is defined in code (relative to FILTER_RADIUS). These are the radius of the convolution filter and its total length respectively. Your code should use the threads of a single block so that each thread computes one element of the result image. Your code should:

    i) It commits memory to the device for the input and output image, but also for the filter used in the convolution. This memory should be freed at the end of the execution.
    ii) Transfers the data to the device and the results back to the host.
    iii) Runs the kernel.
    iv) Systematically and carefully checks for runtime errors.

  b) Checks if the result from the GPU agrees with that from the CPU given an acceptable precision (in number of decimal places). If even one element is found to differ outside the tolerable accuracy, the comparison is terminated and an appropriate message is printed.

3) Experimental study:

  a) Up to what image size (always experiment with image sizes that are powers of 2 and larger than the filter length) can you support without a runtime error occurring? Why does this error occur? (*for this query only use filter radius 4*)

  b) For the maximum image size you can support, what is the maximum precision, relative to filter size, that can be supported without introducing comparison errors?

4) Write code that solves the problem observed in step (3a), using multiple square thread blocks organized in a square grid. Up to what image sizes can you now support?

5) Experimental study:

  a) Study and plot the maximum target precision (as number of decimal places) that results in successful comparisons as a function of filter size. What do you notice? Why is this happening;
  b) Measure CPU and GPU execution time as a function of image size and plot them on a graph. For timing measurements experiment with a filter radius of 16 and an image size of up to 16384x16384 (or lower if there's some CPU or GPU limitation that won't let you go that size). In the GPU execution time you must include the execution time for transferring data to and results from the GPU, but not the time for

binding / unbinding the memory for the images and the filter on the device. Use the timing method that seems most appropriate for each platform.

6) Change the type of elements to doubles instead of floats. Repeat the experiments of step (5). What do you notice?

7) Answer the following questions:

a) How many times is each element of the input image and filter read during kernel execution?

b) What is the ratio of memory accesses to floating point operations? Treat multiplications and additions as separate operations and ignore storing the result. Only count reads from GPU global memory as memory accesses.

8) The code you wrote in steps (3a) and (4), as you can easily notice, suffers from the divergence effect. The problem is created by the if which checks if part of the filter is outside the table. Starting from the code of step (4), try using the padding technique to solve the divergence problem within the warps. What is the difference between the times of this question and the times observed in question (5)?

Hint: His gradepadding required depends on the radius of the convolution filter.

Tradition :
You must deliver:

- The code of the steps (2), (4), (6) and (8).
- Report with the answers to the questions (and its resultdeviceQuery).

Method of delivery :
Create one file . tar.gz with the above contents and name <name1>_<AEM1>_<name2>_<AEM2>_lab3.tar.gz. Upload it on time to e-class.

Delivery deadline :
Monday 21/11/2022 23:59.

Observations :

- All development will be fo system inf-mars1 (10.64.82.31) (or on your own machine if CUDA is supported), while the final measurements on csl-artemis (10.64.82.65). The slots reservations concern, as in the previous work, csl-artemis.
- Thecsl-artemis has a Tesla K80 card. The card has 2 GK210 GPU chips (Kepler architecture). inf-mars has 1 GTX690 card, also with 2 chips (also Kepler architecture). Pay attention because the answers to some of the questions will vary!
- Some (very old)GPUs do not support double precision floating point. Take this into account when compiling for step (6). Both K80 and GTX690 support.
- If you work on your own machine and use the same oneGPU to run CUDA code and also to "drive" your screen, the maximum allowable execution time of any kernel may be limited to about 5''. After the time

this system may forcefully terminate the kernel.

- Be sure to release all dynamically allocated memory and doreset the device at the end of your program, regardless of whether it terminated normally or failed an intermediate check and terminated prematurely.