HPC Lab3

Konstantinos Konsoulas
Ioannis Roumpos

20 November 2022

Contents

1 Introduction

In this assignment we were given the code that a 2D filter (kernel) is going to implement in a 2D matrix, which we can assume is an image. In more detail, the goal of this Lab is to get familiar with CUDA environment, the problems we may face using different grid architectures and the accuracy we can have in our computations using different data types.

2 Analysis

2.1 Device Query

In the Appendix at the end of this report we can see the characteristics of the system we run our experiments. This, will be very helpful in the next questions we will answer.

2.2 Single Block

The first part of this assignment was to write code which will use the threads of a single block in order to compute each pixel of the resulting image (matrix). In addition, a predefined tolerance was given to us to check the accuracy between GPU and CPU resulting images.

Having implemented the above we experimented and observed that for a given radius of the filter the maximum size of the image is 32x32. We attribute this observation to the fact that ,according to device query, the maximum number of threads that a single block can handle is 1024. Since, the height and width are equal in our code the maximum height and width is 32.

In the following graph, we see maximum accuracy in number of decimal digits in regard with the filter number.

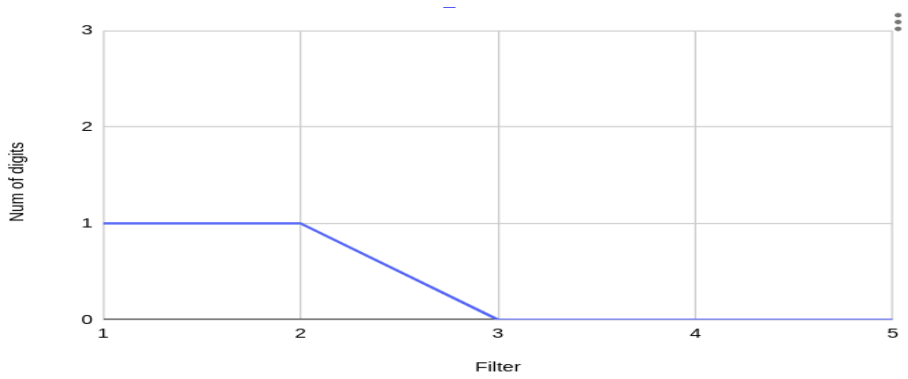

Figure 1: Number of decimal digits in regard with the size of filter.

As we observe for filter radius of 1 and 2 the accuracy is one decimal digit and zero for the rest.

2.3 Multiple Blocks in a Grid

Images of 1024 size can easily be supported by the CPU. So, larger images should be supported by the GPU in order to see difference in the computation time and to achieve that we implement a 2D grid of blocks. In this architecture we can support a larger image. In more detail, our calculatios depends on the available number of threads and memory we have. In terms of memory according to Device Query we have $M = 11997020160$ Bytes, M/sizeof(float) = M/4 = 2999255040 = numberOfFloats, floor(sqrt(numberOfFloats)) = 54765 = imageWidth = imageHeight. In terms of threads, with the help of the grid we can utilize much more threads in number than the available memory. So the maximum image size is 54765x54765.

As in single block the following figure shows the relationship between number of digits accuracy and the size of the filter. With the increment of the filter the error is increased.This is happening because in IEEE 754 the commutative property does not apply every and with the increament of the filter the number of computations of float variables increases the error.

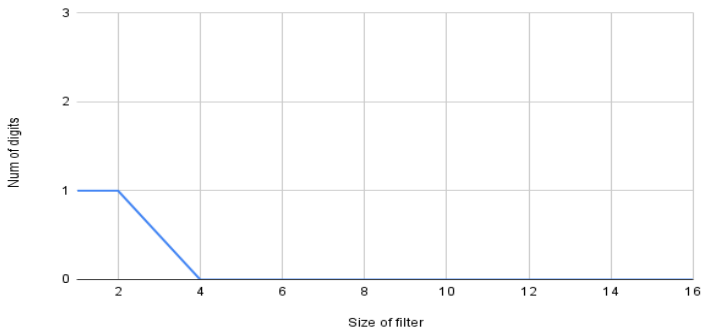

Figure 2

In order to observe a difference between CPU and GPU we measured the computation time for both of them but we did not include the duration of memory allocations on the main memory.

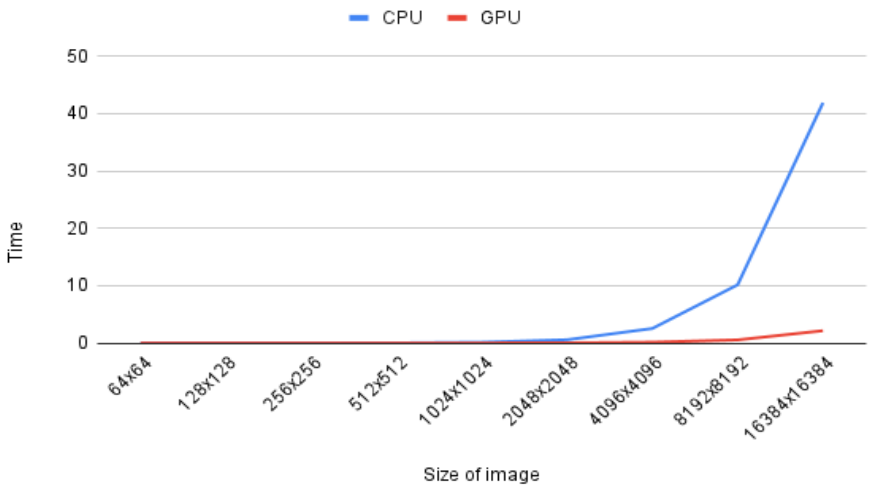

Figure 3: CPU vs GPU

As it can be observed from the graph above GPU outperforms CPU in every single image size. which is expected since the CPU code is completely sequential

and also because GPU handles arithmetic operations better than CPU.

2.4 Change Data Type

In the code, we have written and got our results, we store the data in floats variables. If we change the data type from float to double we increase the accuracy and we observe that is constantly at 0.00005 as we expected. This is explained because doubles are twice the size compared with floats.

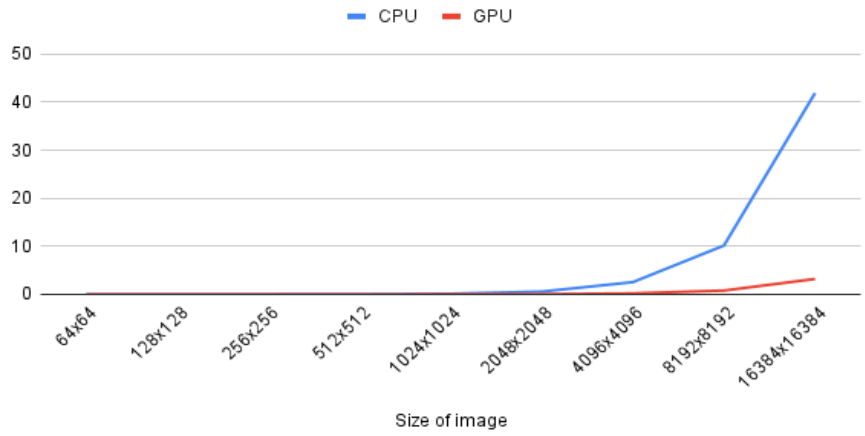

Figure 4: CPU vs GPU Double Data Type

We can observe a small difference with the previous figure to the GPU times as the image gets bigger. That overhead can be attributed to the fact that doubles are "larger" data types and therefore we expect that small difference in computational times between floats and doubles.

2.5 Memory Reads

Here we want to compute how many times each pixel from the input image and the filter are read during the execution of the kernel. We can summarize the total reads to following mathematical expressions.

For **ConvolutionRowGPU:**
$min((filterRadius + 1 + x), (2 * filterRadius + 1))$
For **ConvolutionColumnGPU:**
$min((filterRadius + 1 + y), (2 * filterRadius + 1))$

The sum of the above is the total number of reads in the input image.

It should be noted that x,y refer to element(x,y) of the input image.

For the **filter**, the total number of accesses of an element x of the filter is: $2 * (ImageW - |filterRadius - x|) * N$,where N is the image size.

Regarding the total number of memory accesses we observe that for every two computations,one addition and one multiplication, we have two memory accesses. So, the ratio is $\frac{2}{2}$

2.6 Padding

The divergence phenomenon occurs in our code because warps that execute the code don't follow the same path and execute different set of instructions. This, can be avoided if we remove the conditional statements and add some extra row and columns filled with zero's known as padding.

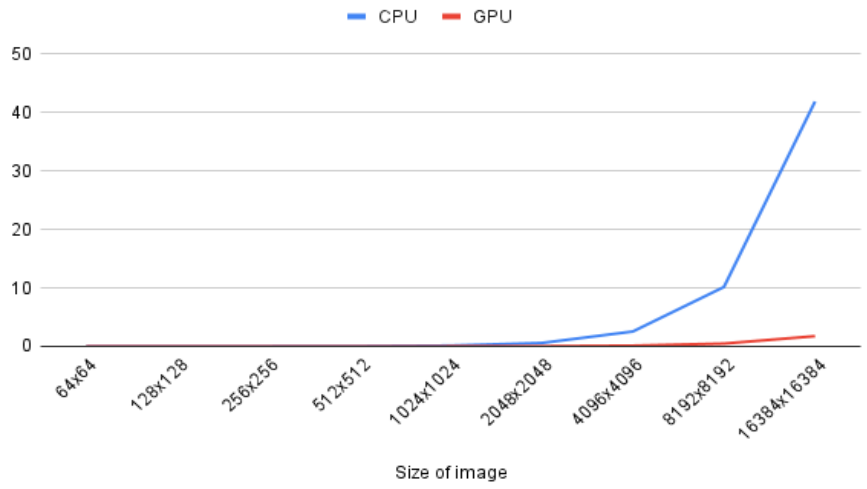

Figure 5: CPU vs GPU with Padding

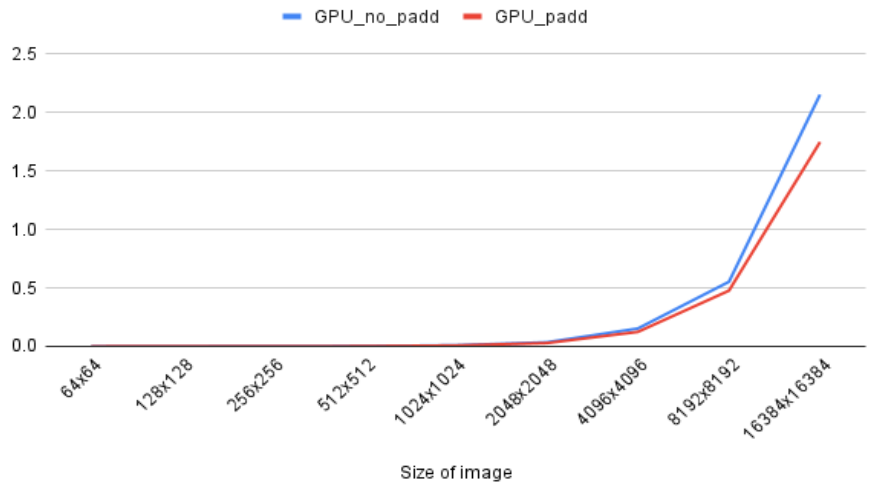

Figure 6: GPU times with and without padding

In the figure above we see the CPU and GPU times with padding to avoid divergence. We observe that the times in GPU are faster than the ones without the padding. This is expected since all the warps execute the same code after the removal of the conditional statements in the code.

3 Contact

Konstantinos Konsoulas(2975): kkonsoulas@uth.gr
Ioannis Roumpos (2980) : iroumpos@uth.gr

Appendix

In this appendix we have the device query result for csl-artemis in which we did the experiments.

Detected 2 CUDA Capable device(s)
Device 0: "Tesla K80"
CUDA Driver Version / Runtime Version 11.4 / 9.0
CUDA Capability Major/Minor version number: 3.7
Total amount of global memory: 11441 MBytes (11997020160 bytes)
(13) Multiprocessors, (192) CUDA Cores/MP: 2496 CUDA Cores
GPU Max Clock rate: 824 MHz (0.82 GHz)
Memory Clock rate: 2505 Mhz
Memory Bus Width: 384-bit
L2 Cache Size: 1572864 bytes

Maximum Texture Dimension Size (x,y,z) 1D=(65536), 2D=(65536, 65536), 3D=(4096, 4096, 4096)
Maximum Layered 1D Texture Size, (num) layers 1D=(16384), 2048 layers
Maximum Layered 2D Texture Size, (num) layers 2D=(16384, 16384), 2048 layers
Total amount of constant memory: 65536 bytes
Total amount of shared memory per block: 49152 bytes
Total number of registers available per block: 65536
Warp size: 32
Maximum number of threads per multiprocessor: 2048
Maximum number of threads per block: 1024
Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
Max dimension size of a grid size (x,y,z): (2147483647, 65535, 65535)
Maximum memory pitch: 2147483647 bytes
Texture alignment: 512 bytes
Concurrent copy and kernel execution: Yes with 2 copy engine(s)
Run time limit on kernels: No
Integrated GPU sharing Host Memory: No
Support host page-locked memory mapping: Yes
Alignment requirement for Surfaces: Yes
Device has ECC support: Enabled
Device supports Unified Addressing (UVA): Yes
Supports Cooperative Kernel Launch: No
Supports MultiDevice Co-op Kernel Launch: No
Device PCI Domain ID / Bus ID / location ID: 0 / 6 / 0
Compute Mode:
¡ Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) ¿

Device 1: "Tesla K80"
CUDA Driver Version / Runtime Version 11.4 / 9.0
CUDA Capability Major/Minor version number: 3.7
Total amount of global memory: 11441 MBytes (11997020160 bytes)
(13) Multiprocessors, (192) CUDA Cores/MP: 2496 CUDA Cores
GPU Max Clock rate: 824 MHz (0.82 GHz)
Memory Clock rate: 2505 Mhz
Memory Bus Width: 384-bit
L2 Cache Size: 1572864 bytes
Maximum Texture Dimension Size (x,y,z) 1D=(65536), 2D=(65536, 65536), 3D=(4096, 4096, 4096)
Maximum Layered 1D Texture Size, (num) layers 1D=(16384), 2048 layers
Maximum Layered 2D Texture Size, (num) layers 2D=(16384, 16384), 2048 layers
Total amount of constant memory: 65536 bytes
Total amount of shared memory per block: 49152 bytes
Total number of registers available per block: 65536

Warp size: 32
Maximum number of threads per multiprocessor: 2048
Maximum number of threads per block: 1024
Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
Max dimension size of a grid size (x,y,z): (2147483647, 65535, 65535)
Maximum memory pitch: 2147483647 bytes
Texture alignment: 512 bytes
Concurrent copy and kernel execution: Yes with 2 copy engine(s)
Run time limit on kernels: No
Integrated GPU sharing Host Memory: No
Support host page-locked memory mapping: Yes
Alignment requirement for Surfaces: Yes
Device has ECC support: Enabled
Device supports Unified Addressing (UVA): Yes
Supports Cooperative Kernel Launch: No
Supports MultiDevice Co-op Kernel Launch: No
Device PCI Domain ID / Bus ID / location ID: 0 / 7 / 0
Compute Mode: ¡ Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) ¿
¿ Peer access from Tesla K80 (GPU0) -¿ Tesla K80 (GPU1) : Yes
¿ Peer access from Tesla K80 (GPU1) -¿ Tesla K80 (GPU0) : Yes

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 11.4, CUDA Runtime Version = 9.0, NumDevs = 2
Result = PASS