Parallel Computing

Lab 4 - Convolution

Name	Sec	B.N	Code
Basma Elhoseny	1	16	9202381
Sara Mohamed Hossam	1	29	9202618

The observed results were tested on output block dimensions of 16 x 16.

		Kernel 1		Kernel 2		Kernel 3	
		Constant	Normal	Constant	Normal	Constant	Normal
512x256 images	Averaging Filter 3x3	213.89us	232.83us	210.91us	241.12us	220.87us	240.61us
	Averaging Filter 9x9	1.6298ms	1.8487ms	1.2887ms	1.6507ms	1.7210ms	1.5615ms
2000x1500 images	Averaging Filter 3x3	4.5195ms	5.1809ms	4.7189ms	5.8547ms	4.8305ms	5.3571ms
	Averaging Filter 9x9	33.984ms	39.688ms	26.769ms	35.751ms	32.113ms	37.574ms

Notes & conclusions:

- Increasing the filter or the input image dimensions results in increasing the overall kernel time, despite the used technique.

- In case of increasing the filter dimensions, k2 & k3 are both severely affected, but k2 responds to that increase better, as the block dimensions match the input tile, providing an increased number of threads, each loading their *only* shared memory element, in contrast to k3, where each thread loads more than one shared memory element, causing an overhead and slowing down the process.

- Using constant memory increases the performance of a kernel, almost all the time, since the filter is cached.

- Kernel 2, where the block dimension matches the input tile, provides an overall better performance.

Kernel (1)

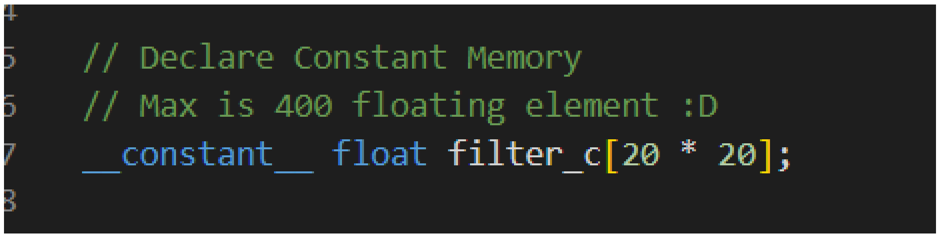
First, we have Tried to use the 3Darray to CUDA, but we have faced some **challenges forced us to use the linearized version again:**

1. The Kernel worked well for small image, but in case of large images the compilation passed but when running the .out file nothing is executed [It Fails in step of Memory allocation]
   We found out that this static allocation is done in the stack which is small that can't be used to hold all this data.

   Unsigned char image [HEIGHT][WIDTH] [CHANNELS]

2. We need the Image Dimensions to be available at compilation which isn't partial instead we need to take the image dimensions form the first image read from the input directory.

   For Constant Memory Allocation we first use 2D Static Indexing but still same problem to use different filters we need to change the filter dim in the code and recompile which is time consuming, so we used Static allocation with max limit 400.

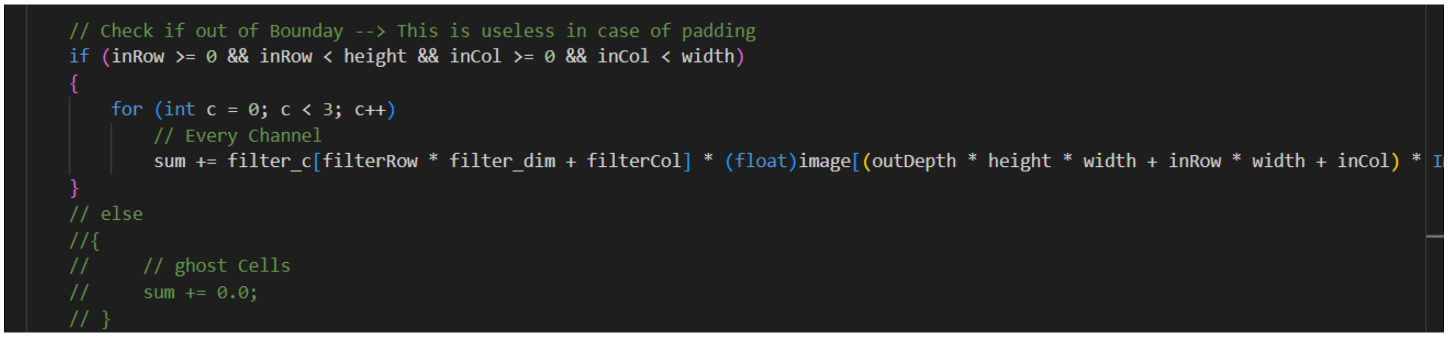


```
// Declare Constant Memory
// Max is 400 floating element :D
__constant__ float filter_c[20 * 20];
```

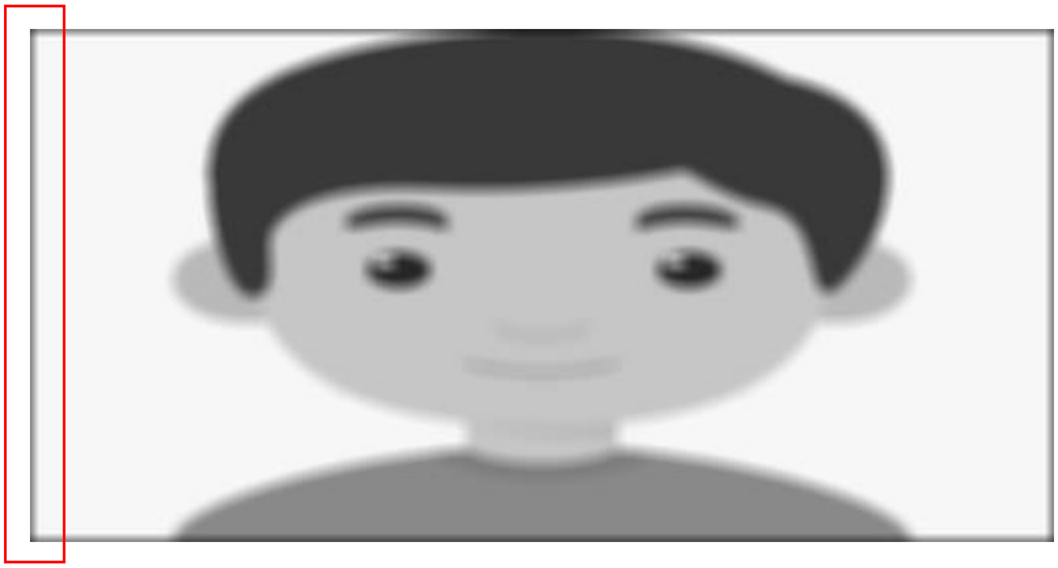
Padding:

Without Padding we just Added Boundary Conditions for out-of-range input. The output was the same size as input because initially we allocated with this, but we got the outer frame to be zero [Default Value]

We added Ghost Cells [No Thing to Be Done]



```
// Check if out of Bounday --> This is useless in case of padding
if (inRow >= 0 && inRow < height && inCol >= 0 && inCol < width)
{
    for (int c = 0; c < 3; c++)
        // Every Channel
        sum += filter_c[filterRow * filter_dim + filterCol] * (float)image[(outDepth * height * width + inRow * width + inCol) * I
}
// else
//{
//     // ghost Cells
//     sum += 0.0;
// }
```



No Black Border 😊 😊 Mask is Grey [Averaged]

Same results applied to Kernel 2 and Kernel 3.