Box Blur

a Boxblur (also known as a box linear filter) is a spatial domain linear filter in which each pixel in the resulting image has a value equal to the average value of its neighboring pixels in the input image. It is a form of low-pass ("blurring") filter. A 3 by 3 box blur ("radius 1")

The following is the results and comparisons of multiple implementations of the Box Blur filter in C++ and CUDA

The implementation

- This filter takes a certain radius as a parameter, which it then uses to iterate through a pixel's neighbors in an r^2 area given that r is the radius, and then sum them into an output pixel, which gives the blur effect.

- This radius r must be an odd number and greater than or equal to 3. $(r \geq 3;\ mod(r,\ 2) \neq 0)$

- When using CUDA, 3 kernels are launched, one on each channel. This is due to the fact that we're working with RGB images.

C++ implementation

We have made two versions of Box Blur in C++, a simple version with no parallelization and another using
OpenMP API parallelization directive.

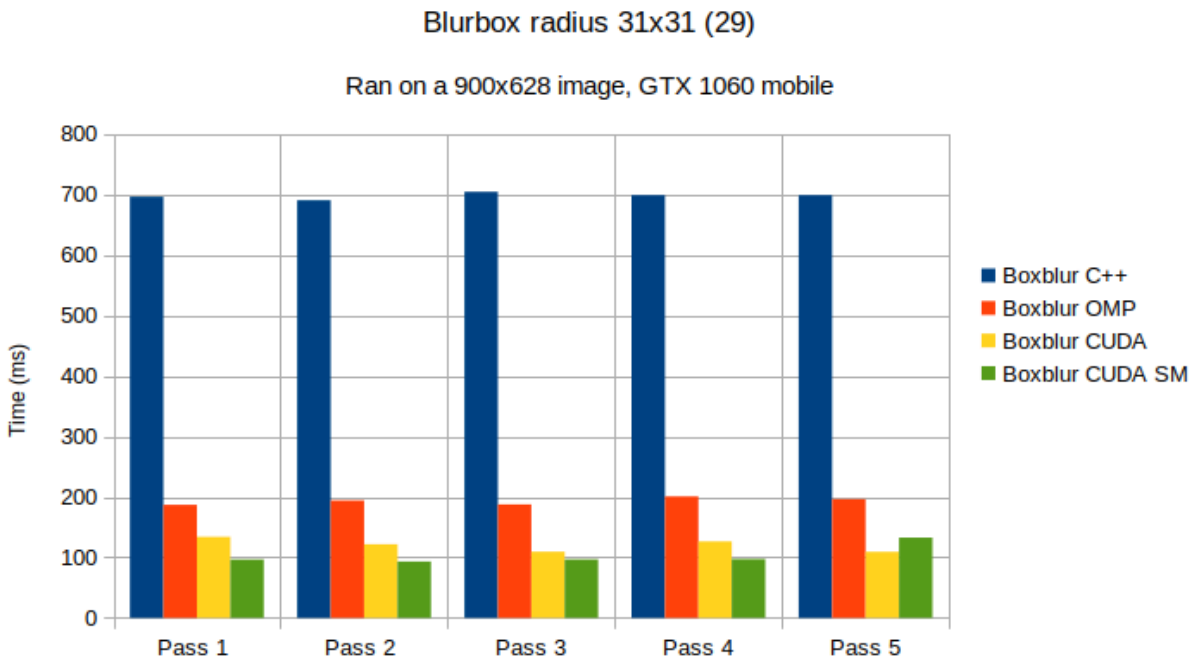
- Using parallelization would surely decrease the computation times by a lot, especially on big samples. As the more threads the program can utilize, the faster it could compute.
- The single-threaded version of the C++ algorithm is expected to be the slowest, especially given a large image.

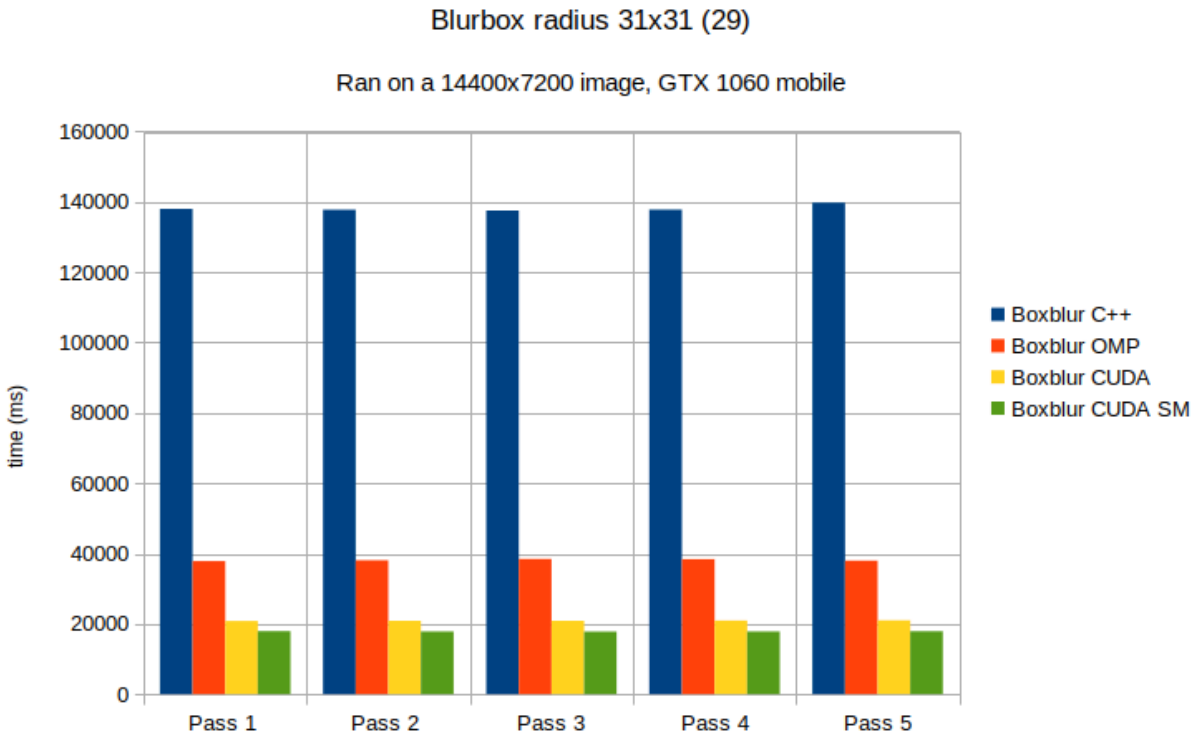
CUDA implementation

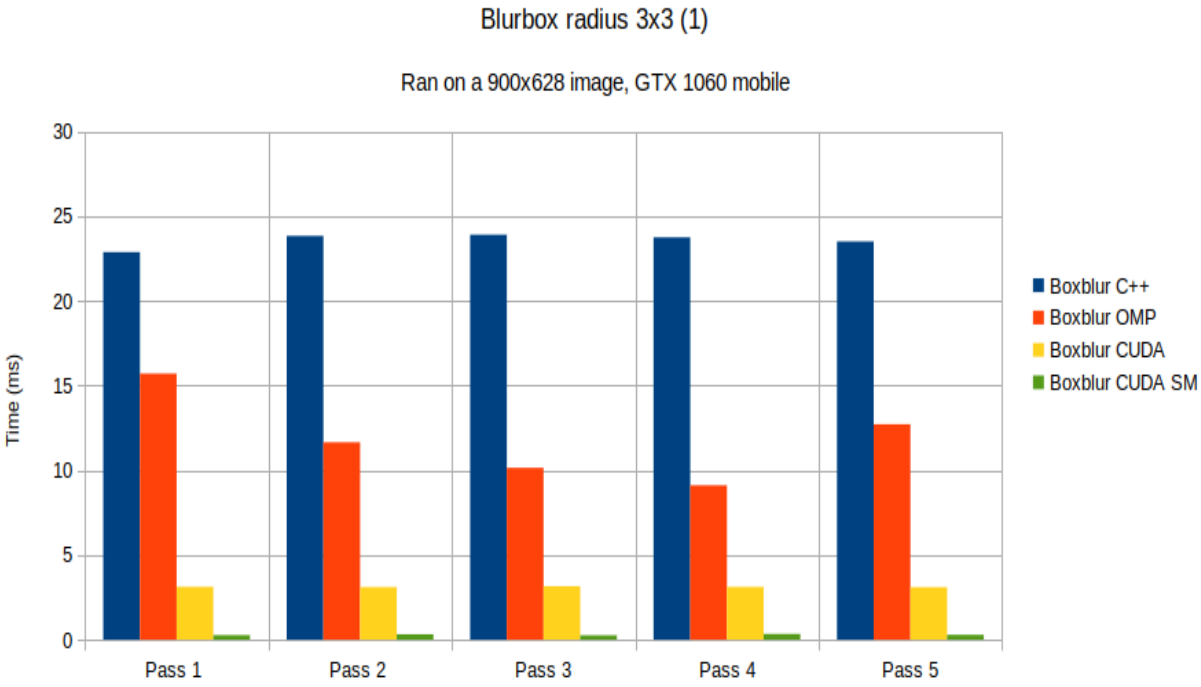
We have also made two implementations in CUDA, one that only uses a basic kernel **(global memory only)** and another that uses ***shared memory***

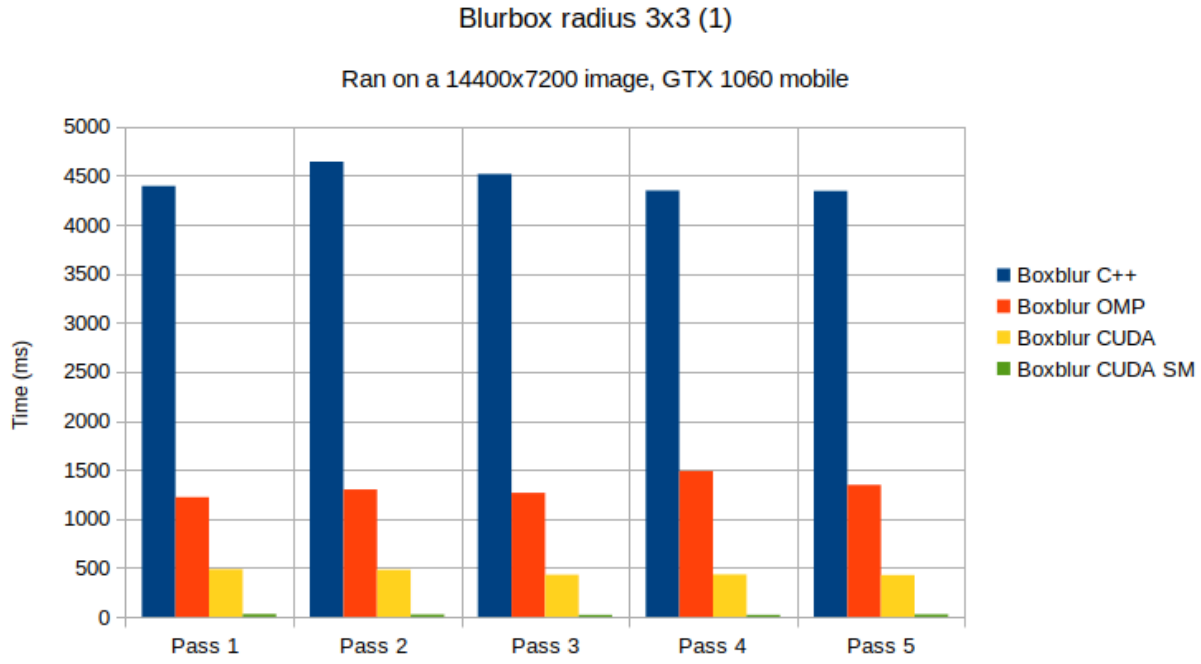
- Using a basic kernel is expected to be a big improvement over both the C++ implementations, or close to the multi-threaded version.
- **Shared memory** is a powerful utility as it provides the same read/write speeds as a register, thus we expect this version to be fastest, however, - *depending on the graphics card model* -, shared memory is rather limited, so using large samples and radii would - *in theory* - **negatively impact** the computation times; as when shared memory is saturated, CUDA overflows to the ***slow*** global memory instead.

Analysis and Results.









Criteria & Setup

- Sample size: **(14400x7200, 900x628)**
- Box blur radius (neighborhood): **(3x3, 31x31)**
- Graphics Card: **6GB VRAM GTX 1060 Mobile.**

- **Shared memory** block size: **32x32**

As we have expected, The single-threaded C++ implementation is the slowest in all 4 scenarios, and in all the 5 passes of each one. However, the **OMP version** is a huge leap in terms of time & performance, as it is **3.8x times faster,** especially in big samples & radii.

The CUDA implementations had varying results on each scenario.

- We could see the shared memory completed the box blur almost instantly on the **14400x7200 sample with radius 3**. It was exactly **17x times faster - finished in 20 milliseconds -**
- In contrast, **increasing** the radius on the **same** sample size, **minimized** the difference between the shared and global versions of the implementations, the shared was still faster nonetheless, but only by about **1.4x times**.

- We suspect that there wasn't enough shared memory for the whole image or that there were **bank conflicts** (which is two (or multiple) threads trying to access the same memory at the same time).
- **Bank conflicts** could be resolved using Streams, but we haven't restored to such a solution given the utter complexity given that this filter depends & uses 3 RGB channels.
- We could see based on the **2nd graph** that when given a small sample with a big radius, the time drops significantly, and from this we could conclude that the criterion most impactful on time is the **sample resolution** as the GPU has to process more pixels, specifically it has to process around 103 million pixels in the big sample, and only about 565,000 pixels, naturally, this big difference results in large differences between times. However in terms of average performance, time between the 2 different implementations of CUDA is rather insignificant.

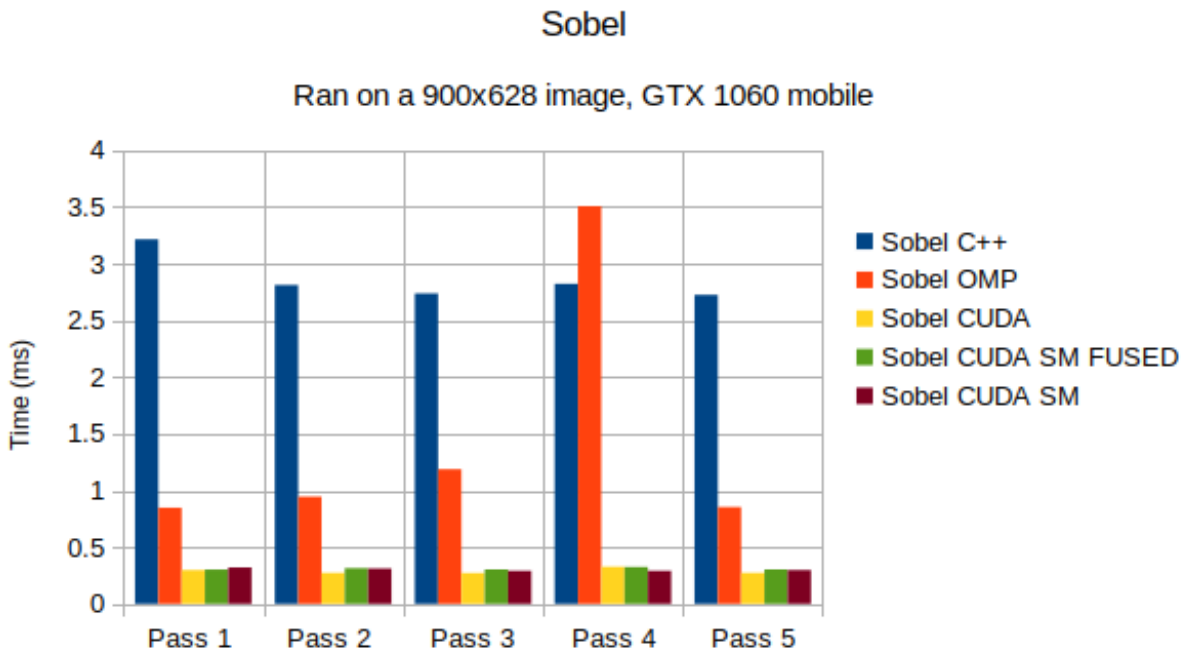
Difficulties encountered when implementing Box Blur

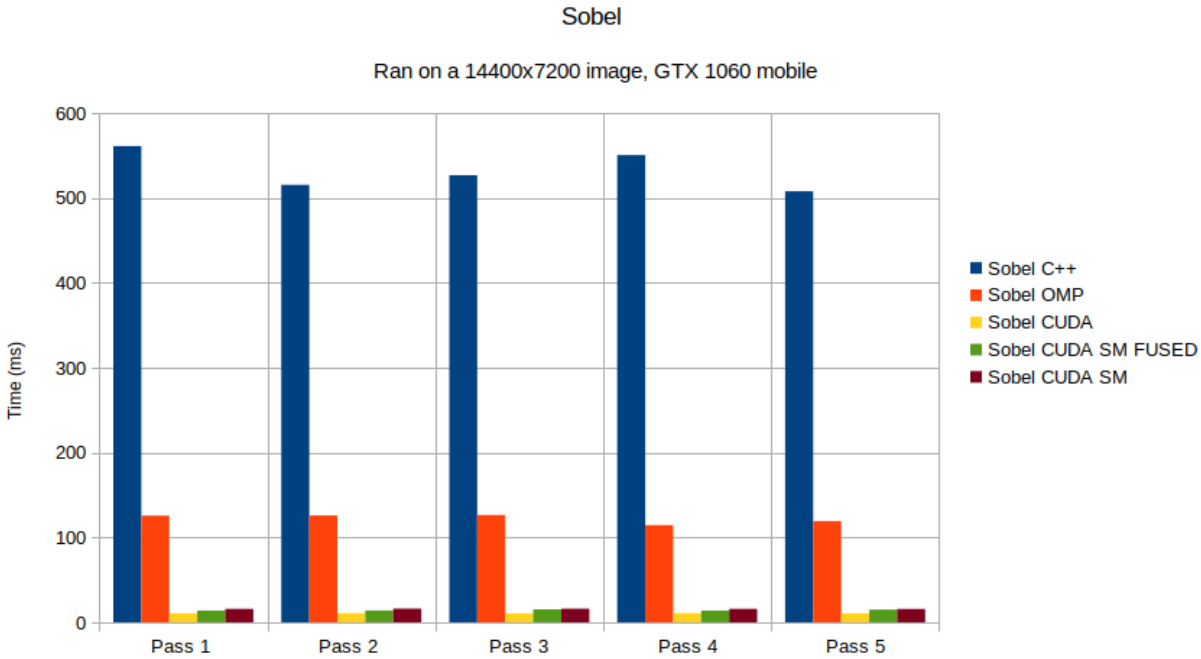
- We had a difficult time implementing shared memory into our algorithm, as we couldn't refer to any examples or documentations as most - if not all - used only one channel (grayscale). This made us resort to improvising by executing the kernel on each channel separately, this approach should by

no means affect performance neither negatively or positively, as the same is happening in the global version, only there we used a for-loop instead.
- Shared memory also introduced a new limitation on our filter, it capped our radius to 31 by also capping the mask size to $32 * 32$.

Sobel (Bonus)

Sobel is an edge-detection filter that applies a convolution mask (kernel) to the neighbors of a pixel.





- Oddly enough the performance is very close between the different CUDA implementations for both samples albeit faster, when it comes to time on a smaller sample. The extreme speed of the kernels could be explained by the fact that sobel depends on the Grayscale filter which also only uses a single channel