* Increasing blur radius = sharper colours.
* Increasing number of blurs does nothing. Why three?
* 3 byte vs. 4 byte for rgb?

Number of times blurred = 3

Blur Radius = 5

rgb

0.01993644

0.00883577

0.01902573

0.00887078

0.01905387

rgba

0.02198468

0.02201773

0.02141553

0.00942409

0.01966447

# GPGPU Assessment 2

## Introduction

**Talk about the project and approach**

The problem this project aims to tackle was speeding up an image sharping algorithm using the OpenCL API.

## Why OpenCL

OpenCL was chosen to take on this approach. Many other candidates existed, including; CUDA, Thrust, AMP, and SYCL. However, OpenCL was chosen for a number of reasons. It is a standard, and is widely used, and a lot of resources exist online for using it.

It is also much more portable than CUDA, which will only work on NVIDIA GPU. By contrast, OpenCL will work on NVIDIA, AMD, and Intel GPUs, which are the three main ones on the desktop platform.

**Because OpenCL is much more widely used than Thrust, AMP, SYCL, and there are a lot more resources for learning it. The C API was used over the C++ one in order to make the code more portable… or something?**

## Tools used

Various tools were used throughout the development of this project. These include; Sublime Text 3, Microsoft Visual Studio 2015, OpenCL, Microsoft Word 2013, Git, and Github.

**Write about GPU and CPU comparisons with stats.**

## Blur

This first part of the project which I moved onto the GPU was the blurring of an image. Because the blur algorithm used **(What’s this called)** works on individual pixels, it was a very good candidate to get paralysed.

First, the blur code will allocate memory on the GPU to store the original image. Then it will copy the memory from the CPU to the GPU. The blur then allocates two extra buffers, which are used to store the blurred data. For however many blurs the user wants (the default is three), the program will calculate the blur and write it into the first buffer. Then it will calculate a blur on the first buffer and write it into the second. Then it calculates a blur on the second buffer and write it back into the first. Having these buffers swap allows there to be an infinite number of blurs performed while keeping the number of memory allocates on the GPU at two.

Once the blur is done, the passed the final blurred image into the sharpening code.

## Sharpening

The sharping code works by subtracting the original image from the sharpened image pixel-by-pixel. This means that, for each pixel, the Red, Green, and Blue components will be subtracted.

The sharpening code was moved to the GPU much more directly than the blurring code. The sharpening function was changed into an OpenCL kernel function. To call the function, two simple utility functions were created; *set\_argument\_helper* and *run\_kernel*. The first of these is just used to abstract away passing parameters to OpenCL. It increments a global counter variable, so that the parameters can be passed sequentially without the user having to worry about it. The second function is used to launch a kernel. It also sets the global counter back to zero.

## Increasing Blur Radius

**Talk about this made the image brighter.**

## Increasing number of times image is blurred

**Talk about how this did nothing**

## Conclusion

**What Did I learn??**