There's not much to show graphically for this report. The basic idea is that everything runs in parallel, so there's no runtime speedup. Everything else below explains the project and how to use my program.

Overview

The idea of the project is to take a bmp file (pixels arranges in [RRR...RGGGB...BBB] order) and convert it to a greyscale image. We do so by partitioning the image into blocks and grids. Each block contains a 2D array of pixels (which will translate to 2D array of cores in CUDA). Each grid will correspond to a 2D array of blocks.

Once we partition as mentioned above, we pass the image to memory in the GPU and call a kernel function that will perform the greyscale. To do this we must get the RGB values for each pixel and set all 3 values the following value (R*0.3) + (G*0.59) + (B*0.11). The only remaining tricky part is how to get the RGB values. Once we know how many pixels are in the image (call this *length*) and we have the index of the R value for a pixel in the array, we can easily get the other values. We can add (R_index+*length)* to get G, and add (R_index+2**length)* to get B.

Runtime

We can run "make benchmarks" and this will run our program with different values of the num blocks. When we do this, we can see that the time elapsed doesn't really improve much (if at all). For example, the slowest when I ran it was 0.124932s. Every other run (with blocks between 2-32) completed with time between 0.07s and 0.10s.

Therefore, changing the number of blocks doesn't lead to a clear improvement of performance (in terms of time). This makes sense because when we create our block of threads and grid of blocks, the total number of threads is the same no matter how we partition between blocks/grids. Increasing block dimensions reduces the grid dimensions. Every time, we are creating exactly the same number of threads (which is equal to the total number of pixels). Hence, each pixel is processed by a separate thread in parallel, leading to similar runtimes. Any speedup/slowdown could be attributed to scheduling discrepancies.

Usage:

Make can be run in the /src folder of the package. The program takes about 10 seconds to compile since CImg is rather large in comparison. Once the program is compiled, there are 2 make commands:

- make *benchmark*
  Run the program without different number of blocks each time and display the time elapsed. Number of blocks will vary between 1-32.
- make *run*
  Run the program a single time and display the time elapsed (by default uses block size=1).

We can pass optional args to the makefile for running the program using variable RUN_ARGS. For example, we can pass in the optional y flag (display the image before the greyscale) and the optional z flag (display the image after the greyscale) using the following command:

```
make run RUN_ARGS="-y -z"
```

A full list of commands can be printed using optional flag -h.