Assignment III: Advanced CUDA

Amirhossein Namazi, Calin Capitanu

November 23, 2021

Exercise 1

CUDA Edge Detector using shared memory

1. Explain how the mapping of GPU thread and thread blocks (which is already implemented for you in the code) is working.

   The thread blocks are spread into a grid, since the processing of each pixel is easier to be sent out and understood in an actual 2D format. There is a defined BLOCK_SIZE of 16 threads (per block), and then the size of the image is divided into smaller blocks, thus creating a grid. One could easily visualize this as if pixels are grouped into small squares (the blocks) and each pixel is considered a thread inside this thread block.

2. Explain why shared memory can (theoretically) improve performance.

   On short, shared memory could theoretically increase the performance due to the fact that the bandwidth inside the device (GPU) is way higher than in the global memory (between CPU and GPU). However, depending on the amount of processing, it could actually not be that beneficial. If threads do not need to share much memory (for example only 1 interconnected execution), the performance could as well not be highly improved. Most of the time however, the performance is better since threads share a block of memory inside the same thread block and do not need to access the host (CPU) in order to get data from an adjacent thread.

3. Explain why the resulting image looks like a "grid" when the kernel is simply copying in pixels to the shared block. Explain how this is solved and what are the cases.

   Since the BLOCK_SIZE_SH is set with 2 more columns and 2 more rows, when the gpu_applyFilter function is called, the whole block is sent to the function and when one arrives at the end of the block, there are these two more columns and rows that do not have any data, thus they are undefined and get unexpected results. That is due to the fact that the filter function actually does work with a 3x3 matrix, thus when arriving at the last rows and columns, there is nothing left to work with for these pixels, thus the need for extra columns and rows, in order to compute the pixel with the 3x3 filter.

4. There are several images of different sizes in the image folder. Try running the program on them and report how their execution time relates to file sizes.

   The short answer is yes. The images size does matter with computation time. In fact the image "hw.bmp" and "nyc.bmp" took the longest. It is also pretty intuitive that more images will take longer (unless there are enough available threads on the GPU to run all of them in parallel). "nyc.bmp" took around 25 ms for each run and "hk.bmp" took around 13-14 ms for each part, while the other took 8-9 ms. The results are displayed in the plot below.

Pinned and Managed Memory

Exercise 3

CUDA Streams/Asynchronous Copy - Particle Batching

Bonus Exercise

CUDA Libraries - cuBLAS