GPU Assignment: Image processing

Stefano Sandonà

Vrije Universiteit Amsterdam, Holland

1 GPUs: NVIDIA GTX480

The aim of this assignement was to learn how to use many-core accelerators, GPUs in this particular case, to parallelize data-intensive code. All the implementations were written for the **NVIDIA GTX480**, using CUDA, a parallel computing platform and programming model invented by NVIDIA. Programming with CUDA, there is a straightforward mapping onto hardware, for this reason it is necessary to study the available HW before start developing an application. The main characteristics and the limits of the given accelerator, are showed in Table 1.

Table 1: NVIDIA GTX480 Specifications

Microarchitecture	Fermi
Compute capability (version)	2.0
Maximum dimensionality of grid of thread blocks	3
Maximum x-dimension of a grid of thread blocks	65535
Maximum y-, or z-dimension of a grid of thread blocks	65535
Maximum dimensionality of thread block	3
Maximum x- or y-dimension of a block	1024
Maximum number of threads per block	1024
Cores per SM (warp size)	32
SM	15
Cores	480 (32 * 15)
Maximum number of resident blocks per multiprocessor	8
Maximum number of resident warps per multiprocessor	48
Maximum number of resident threads per multiprocessor	1536 (48 * 32)
Number of 32-bit registers per multiprocessor	32K
Maximum amount of shared memory per multiprocessor	48K

2 CImg

The image processing library used in this project was CImg, a small, modern and open-source toolkit developed for C++. CImg implements the RGB color model, an additive color model in which red, green, and blue light are added together in various ways to reproduce a broad array of colors. Each colored image of size $N\ x\ M$ is composed by three parts (R,G,B) of the same size, so that $N\ x\ M\ x\ 3$ values are necessary to define an image. The Figure 1 shows an example of image composition.

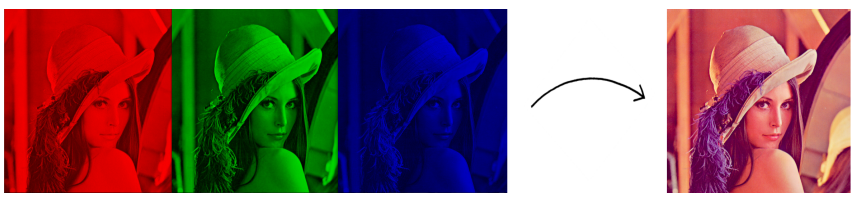

Figure 1: RGB model

3 The processing flow

Using CUDA there are two parts of the code: the device code, or GPU code, or Kernel, that is a sequential program write for one thread and execute for all and the HOST code, or CPU code, that is used to instantiate the grid, run the kernel, manage the memory. Figure 2 shows the processing flow of a CUDA application. In the particular case of image processig, everything starts from the CPU, that store the image from a file into a local buffer, allocates IN and OUT buffers on the GPU (*cudaMalloc*) and copy the image into the GPU's IN buffer (*cudaMemCpy*). After that, the CPU launches the GPU kernel with a defined grid configuration (*kernel_function «gridDim, blockDim»(params)*), that is executed by the GPU following the SIMT (Single Instruction, Multiple Threads) NVIDIA model. The threads are executed in parallel in each core, and they read the assigned part of IN data and generates the assigned part of OUT data. At the end, the results are copied out back to the CPU (*cudaMemCpy*) and the image is written to a file by the CPU.

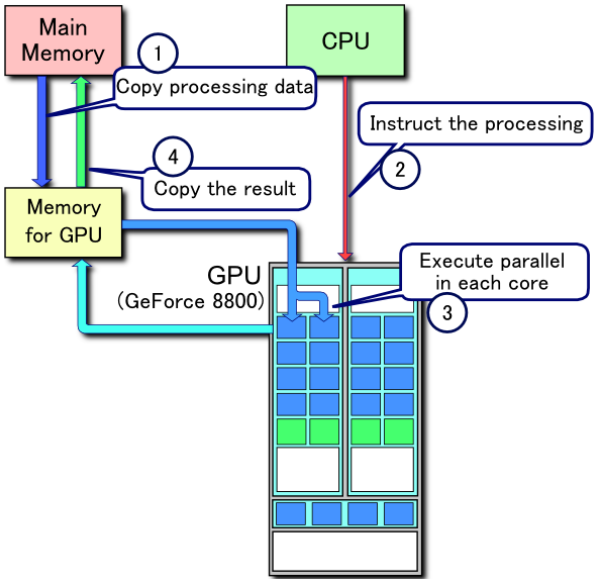

Figure 2: CUDA processing flow

4 Algorithm 1: Grayscale Conversion and Darkening

5 Algorithm 2: Histogram Computation

6 Algorithm 3: Smoothing