Subject-Part1-en.md

Assignment Part One: Obfuscation

Intro

Obfuscation is the art making something hard to understand at the first sight. In this project, someone divided an image into two parts to make it impossible to understand. You goal is to recombine them to find out what was the original image.

Go in the project *1-Obfuscation* and check the images in *data*. In Linux, you can open them from the terminal with `eog file.tif`.

You will have to sum the two images to get the resulting picture.

Please read the subject entirely before writing any code!

Code organization

- CMakeLists.txt -- the build system for the project
- data -- input images
  - big_frag_1.tif
  - big_frag_2.tif
  - small_frag_1.tif
  - small_frag_2.tif
- include
  - obfuscate.hpp -- header file containing some usefull macros
- src
  - main.cpp -- basic structure of the program, you should not need to modify it
  - obfuscate.cu -- **here is where you should work**
  - reference.cpp -- contains reference implementation, comparison function...

Build the code

Just like *deviceQuery*, this is a cmake project. Once your are in *1-Obfuscation*, create and cd to a directory named *build*. Then run `cmake ..` and `make`. Every time you do make change in your code, you will only need to run `make` then `./obfuscation exercise_1` to run the latest version.

CUDA firsts steps

In obfuscate.cu, you will have to implement the basic framework of any CUDA program:

1. Allocate buffer on the device (GPU) memory.
2. Copy input data to the GPU
3. Launch your kernel
4. Copy output data back to the CPU
5. Free the memory you allocated

Some code is already there to help you. In this exercise, you are expected to run your kernel within a **single** block. Just make it the appropriate size for your data.

Here, the image is reprensented as a contiguous array of unsigned bytes (grayscale). You just need to sum the two input images and store the result in the output image.

Feel free to use *reference.cpp* as an example for your kernel implementation. If you need details about a particular CUDA API, some documentation in also included at the root of this project. Just run `tar xzf documentation.tar.gz` to decompress it.

Questions

1. How much time does your CUDA takes? What fraction of this time is used for memory management?
2. Compare with the time taken by the CPU implementation.

Obfuscation (Part II)

They are a few changes with respect to part I:

- the image is now RGB, stored on 32 bit-integers. Use the macros in obfuscate.hpp for your convenience.
- the image is now too large to fit into a single block. You will need a strategy to divide your problem into different blocks.

Run with `./obfuscation exercise_2`

Questions

Same questions as before:

1. How much time does your CUDA takes? What fraction of this time is used for memory management?
2. Compare to the time taken by the CPU implementation.

And a new one: 3. What is the influence of the block/grid division of the problem on the runtime? For which block/grid size did you find the best execution time? Why is that the case?