Programming Massively Parallel Processors Exercise 2

TECHNISCHE UNIVERSITÄT DARMSTADT

TU Darmstadt, WS 2014
André Schulz <andre.schulz@gris.tu-darmstadt.de>

Introduction

Your task is to implement a Gaussian filter to blur images. The implementation will consist of several CUDA kernels in order to get accustomed with the performance difference of the GPU's different memory types and caches.

Prerequisites

If you want to work with your own computer, you will need a CUDA-capable graphics card and the following software:

- C++ compiler (GCC or Visual C++)
- CMake 2.8 or higher (`http://www.cmake.org`)
- CUDA Toolkit 5.0 or higher (`https://developer.nvidia.com/cuda-downloads`)

Task 1 - CPU

One way to blur an image is by convolving it with a Gaussian filter kernel. Gaussian filter kernels are rotationally symmetric which allows separation of the 2D convolution operation into two 1D convolutions, horizontal and vertical. For a more thorough description of separable convolution please refer to the whitepaper of the `convolutionSeparable` CUDA sample which is also available online on `http://docs.nvidia.com` in the `CUDA Samples` section.

A CPU implementation of the two convolution operations is given in `convolution_cpu.{cpp,h}`. Additionally, `ppm.{cpp,h}` contains a class which can load and save PPM images. For generating a Gaussian filter kernel to use with the convolution you can use the supplied `initKernelGaussian1D` function in `common.{cpp,h}`.

Your task is to create a CMake project named `gaussFilter` which creates a binary of the same name. Then implement Gaussian blur by using the given code to load an image, generate a Gaussian filter kernel, apply Gaussian blur to the image and save the blurred image to a file named `out_cpu.ppm`. Your program should take two command-line parameters. The first one is the file name of the image that should be blurred and the second is the filter kernel size.

Task 2 - Global memory

Write a GPU implementation of the horizontal and vertical 1D convolution which works on a `PPMImage`. For this task, you should only use global memory in your implementation. The `PPMImage` class represents the pixels of an image in 4 bytes (1 byte per color channel, 4 color channels (RGBA)). The convolution should only be applied to the RGB color channels; you can ignore the alpha channel. Your implementation should be able to handle different filter kernel sizes. Finally, save the resulting image to a file named `out_gpu_gmem.ppm`.
Note: Be careful to handle out-of-bounds memory accesses at the borders of the image by clamping the accesses to the nearest valid pixel values.

Task 3 - Shared memory

Copy your convolution CUDA kernels from task 2 and modify them to use shared memory for the pixel data. For an idea on how shared memory can help with convolution, see the whitepaper about the `convolutionSeparable` CUDA sample. The blurred image should be written to a file named `out_gpu_smem.ppm`.
Note: Do not implement everything from the whitepaper! A simple implementation which ignores the coalescing rules from old GPUs is completely sufficient.

Programming Massively Parallel Processors - Exercise 2

Task 4 - Constant memory

Copy your convolution CUDA kernels from task 2 and modify them to access the image in global memory and the filter kernel in constant memory. Because the amount of constant memory is determined at compile-time, you can restrict the maximum filter kernel size (e.g. 65). Save the blurred image to a file named `out_gpu_cmem.ppm`.

Task 5 - L1/texture cache

Copy your convolution CUDA kernels from task 2 and modify them to access both pixel data and the filter kernel in global memory via the L1/texture cache by using the `__restrict__` keyword. For details, read chapter B.2.4 in the CUDA Programming Guide. Save the blurred image to a file named `out_gpu_restrict.ppm`.
Note: Usage of the L1/texture cache with the `__restrict__` keyword only has an effect on GPUs with a compute capability of 3.5 or higher. Additionally, you have to tell `nvcc` to compile the CUDA kernels for a specific compute capability by using the `-arch` option. (e.g. `-arch=sm_35`) Passing the option in CMake is done by setting the `CUDA_NVCC_FLAGS` variable: `set(CUDA_NVCC_FLAGS "-arch=sm_35")`

Task 6

Copy your convolution CUDA kernels from task 2 and modify them such that they combine all of the optimizations from the previous tasks: cache pixel data in shared memory by loading it from global memory via the L1/texture cache and access the filter kernel in constant memory. Save the blurred image to a file named `out_gpu_final.ppm`.

Task 7

Observe the runtime difference between each task's implementation by profiling your program using `nvprof` or the NVIDIA Visual Profiler. For details on both profilers refer to the documentation in the CUDA Toolkit or online at `http://docs.nvidia.com` under `Tools` and then `Profiler`.

Modalities

For the exercises, group work is not allowed. The program should compile and run on any workstation in the Linux lab (Windows is not allowed). Please submit your solution (an archive of your project directory) using Moodle. The archive should have the following naming convention: `PMPP14_ex2_name.zip` (substitute "name" with your name). The deadline is Nov 11th, 12:00.