CS 250B: Modern Computer Systems

Lab 2: Stencil codes on CUDA

Due: 2023-12-13

Overview

In this lab, you will take a naïve software implementation of a scientific simulation algorithm, and port it to a General Purpose Graphic Processing Unit (GPGPU) using NVIDIA Compute Unified Device Architecture (CUDA). The target application is the same as in lab 1, stencil codes for two-dimensional heat dissipation simulation.

Provided Material

The provided files are very similar to the ones in lab 1. The CUDA implementation of the stencil kernel should be done in **stencil.cu**, which is again, ***the only file you should modify***. Compilation and execution is almost identical to the version in lab 1, except there is no thread count argument. An example execution will be the following (8192 time steps, starting with init.dat, using the CUDA kernel):

./obj/stencil 8192 init.dat n

For supported GPUs, you can also run the NVIDIA profiling tool nvprof via the following command:

nvprof ./obj/stencil 8192 init.dat n

For those of you who may not own a CUDA-supported GPU, two shell scripts are also included for executing it on the UCI HPC cluster with NVIDIA GeForce GTX 1080 GPUs. The use of this will be described below.

Using The UCI HPC Cluster

For those of you who does not own a CUDA-supported GPU, an option is to use the UCI HPC Cluster. All postgraduate students are granted access to the cluster. You will need to sign up for an account, following the instructions here: https://hpc.oit.uci.edu/signup.html

Once you log in to **hpc.oit.uci.edu**, you can **scp** the lab code to your home directory. After you have the code, you will need to load the CUDA module to compile the code, via the following command:

module load cuda/10.1

You should be able to compile the lab code afterwards.

There are two shell scripts provided in the lab code directory for submitting the CUDA code to the cluster: submit.sh and run.sh. run.sh is the file to be submitted to the cluster, and submit.sh is the script that does it. You can submit your compiled binary to the cluster by running the following command:

bash submit.sh run.sh

The current version of run.sh is giving a very small step count to the kernel, because the single-thread implementation provided is extremely slow. You can slowly increase the number as your performance improves.

The results of the execution will be stored in a set of files in the working directory: run.sh.eXXX…X and run.sh.eXXX…X (for stderr and stdout, respectively). Since it will take a while for your execution requests to be picked up by a free node and finished, you can check the status of your requests by running the following command:

watch "qstat –u [your id]"

Unfortunately, the HPC cluster does not allow nvprof execution, so your optimizations will have to depend mostly on reasoning.

Suggested Approaches

Since CUDA does not support transparent caching, it will be very tricky to implement a cache-oblivious algorithm. So this is not the goal right now.

Instead, focus on a 2-D block of threads copying blocks of data into an explicit shared memory block, and performing a single sweep of the kernel execution. The point of interest will most likely be setting the correct block and shared memory size for the underlying GPU architecture. These parameters can be set in the host-side function stencil_optimized, which calls the GPU-side kernel stencil_cuda.

Keep in mind that unlike the CPU version of the lab, each buffer, temp and temp2, consist of only one time-stamp.

In this version of the code, please don't worry about handling edges. You can just focus on handling cells from (1,1) to (width-1, height-1).

Submission and Grading

Please submit two things:

"***stencil.cu***", and a short documentation describing your implementation (ideally including a schematic), and the performance evaluation.