Task 2: GPU Programming

Introduction

For this task it is necessary to have an Nvidia GPU since we are going to use CUDA API. If your computer has only one GPU, it is important to note that when using the GPU for general purpose programming as well as for visualization (OS), if any serious error occurs or the program requires too many resources, it can cause a "hang" of the system. Therefore, it is generally recommended to use an additional GPU for general purpose programming.

For this reason, or because an Nvidia GPU is not available, in this task we are going to use the Google Colab environment (https://colab.research.google.com/?hl=en) that will allow us to have a remote computer with GPU for free. It is only necessary to have a Gmail account.

The first time we access Google Colab, the following description appears:

“Colab, or "Colaboratory", allows you to write and execute Python in your browser, with

- Zero configuration required
- Access to GPUs free of charge
- Easy sharing

Whether you're a student, a data scientist or an AI researcher, Colab can make your work easier. Watch Introduction to Colab to learn more, or just get started below!.”

In our case we are not going to use Python, although it is possible to program CUDA in Python (PyCUDA - https://documen.tician.de/pycuda/) but we are going to program in C/C++. As we are going to use a notebook-based environment, we can make use of the “NVCC Plugin for Jupyter notebook” that will allow us to use the cells of the notebook to execute CUDA code. We will also show you how to run CUDA in the Collab notebook without the plugin.

Exercise 0: Setup

This exercise will introduce the use of Google Colab for CUDA programming. We provide a basic notebook with all required steps to execute CUDA programs in Colab.

1. Obtain a copy of the notebook Setup_GPU.ipynb.
2. Create a folder in your Google Drive to store the notebook (recommended) and upload it.
3. Doble click on the notebook. You will see the content.

4. Click on "*Connect*". A virtual environment will be assigned to your notebook, including a CPU and a GPU.
5. You can execute OS commands using "!" before the command. For example, the first cell list the CPU model executing "*!lscpu*" and the next command shows the amount of memory available executing "!free -f".
6. The command "*!nvcc –version*" shows the version of the CUDA compiler and "*!nvidia-smi*" shows details about the available GPU. Yes, CUDA is installed by default.
7. If you want to use the "NVCC Plugin for Jupyter notebook" is necessary to install it. You can clone the git (sometimes it does not work) or download a zip file and install it just executing the commands required for that. Then you must load the plugin.
8. The first example is a Hello world in CUDA that can be executed in the cell thanks to the plugin. The second one is a simple add of two numbers. Note that you can write all your CUDA code in one cell and execute the code thanks to the plugin.
9. The next example shows you how to use multiple files for your CUDA program and compile and execute the program without the plugin. Basically, you have to save the files (one per cell) and then compile the files using NVCC compiler. If the compilation completes successfully, you can run the program.
10. Last cells show you how to connect your Google Drive into your Colab notebook to share files (required for exercise 2) using Python.

Exercise 1: Stencil 1D

In this exercise we will work with the stencil 1D algorithm explain in class. Our goal is to compare the GPU vs. CPU performance. Remember that in CUDA you can run CPU code, so you can add the CPU 1D stencil code in the same program or create a different program. To compare performance, use *gettimeofday()* to measure the execution time. In GPU you must include the time to send and receive data to/from the GPU.

1. Use the most optimal stencil 1D GPU kernel from the slides, the one that makes use of blocks + threads + shared memory.
2. Compare the execution time for different values of N (array size): from 100.000 to 1.000.000 in steps of 100.000. Plot the result in a graph. Explain the results.
   - Tip: You can use Python to generate the plot in the notebook.
3. What BLOCK SIZE (number of threads per block) have you used? Do you think it is the most optimal? Explain.

Note: you can plot the speedup instead of the execution time of CPU and GPU.

Exercise 2: Image processing algorithm

In laboratory task 1 part 2 we vectorized the file *greyScale.c*, that applies an image processing algorithm to convert any image to grey scale algorithm. The goal was to process images from a video stream in real time, that is, approximately 30 fps (frames per second). Now we want to use a GPU to convert the images and compare the performance.

1. Implement a GPU version of the program. Basically, you must write a CUDA kernel to compute the algorithm and add the code to send and received the images to/from the GPU.

   a. It is imperative that the program continue to perform the same algorithm, so only changes should be made to the program that do not change the output.
   b. Note: You can process the image as an array (1D) or using a 2D approach.
   c. Important: Independently of the nchannels value get from the image, the algorithm always use nchannels = 4 for dynamic memory allocation (malloc) and to compute the grey scale algorithm.
2. Fill in a table with time and speedup results compared to your manually vectorized CPU code for images of different resolutions (SD, HD, FHD, UHD-4k, UHD-8k). You must include a column with the fps at which the program would process. Discuss the results.
3. Explain how you implement the algorithm to be optimal for GPU.

Material to submit

You must write a report answering the questions proposed in each exercise, plus the requested files. Submit a zip file through Moodle. Check submission date in Moodle (deadline is until 11:59 pm of that date).

- From exercise 0:
  - None
- From exercise 1:
  - Provide the source code of your stencil 1D implementation (CPU and GPU).
  - Plot the results of the experiment and explain them.
  - Answer all the questions in the report.
- From exercise 2:
  - Provide the source code of the GPU version of the code. Explain your solution.
  - Create a table with the results of the experiment and explain them.