An Introduction to HPC and Scientific Computing

CWM, Department of Engineering Science

University of Oxford

Practical 5: Practical examples of CUDA libraries.

This practical will review some of the CUDA libraries and their uses which were discussed in the lecture "An introduction to GPUs and how to use them".

The learning outcomes of this practical are:

- To have some understanding of CUDA libraries and their uses.
- To understand how memory is allocated on the GPU.
- To understand how data is transferred to and from the GPU.

All practical sessions for this course will be carried out on the University's ARCUS-HTC computer. To understand how to use ARCUS-HTC see the introduction video and for more details watch the video demonstrations in lecture 3. As a reminder:

1. Connect to the university VPN (instructions can be found on canvas).
2. For windows: open MobaXterm and open a ssh session.
3. Connect to "oscgate" by:
    a. Setting "*remote host*" to be oscgate.arc.ox.ac.uk
    b. Tick "*specify username*" and set it to teachingXY (where this is the username we have issued you with)
    c. This will open a shell, from here you can connect to "arcus-htc" using ssh as follows:

        ssh –CX teachingXY@arcus-htc

4. Or for mac: ssh -CY teachingxx@oscgate.arc.ox.ac.uk; ssh -CY teachingxx@arcus-htc

Instructions for this practical

1. If you have not done so clone the github repo for this CWM. To do this, at the command prompt type:

    ```
    $ module load  git
    $ git clone https://github.com/wesarmour/CWM-in-HPC-and-Scientific-Computing-2020.git
    ```

    If you have already cloned the repo, pull it again to ensure you are working with the most up-to-date codes:

    ```
    $ module load  git
    $ git pull
    ```

2. Next you should navigate to the examples directory in the prac5 folder:

    ```
    $ cd CWM-in-HPC-and-Scientific-Computing-2020/practicals/prac5/code
    ```

3. When logged into the Arcus-HTC head node, you will need to use the command

   module load gpu/cuda

to be able to compile or codes. For running we will use, similarly as in the practicals before, the SLURM scheduler to put the job in the queue, i.e.:

sbatch <*the_sbatch_script*>

The sbatch script is provided only for the first task. For other tasks you need to copy the sbatch from the first task and edit the file to run your actual code.

Part A

1. Read through the code provided in 'prac5/code/cuFFT/cuFFT_C2C.cu'. The main() function is preparing data which will be Fourier transformed by the GPU. Locate where we set the size of the FFT and number of series we want to transform. Find where we initialize our series which we want to be Fourier transformed. Is there a problem?

2. We have written a function 'Do_FFT_C2C_forward'. It provides an example of how to perform an FFT using cuFFT library. Read through comments in the code. Can we reuse the same cuFFT plan with different data? Where would you place a 'for' loop which would do that? Can we reuse same cuFFT plan if we change number of FFTs we would like to calculate? What about the FFT size?

3. Complete function 'Do_FFT_C2C_inverse_inplace' based on reading through function 'Do_FFT_C2C_forward'. In function 'Do_FFT_C2C_inverse_inplace' we would like to perform inverse and in-place FFT by using cuFFT.

4. Run the program and look what it prints to the console. If you have written your code correctly the difference between input and output should be 0 (or very close to it).

Part B (The second part of this practical is courtesy of Mike Giles)

1. Navigate to the code directory in the prac5 directory.

2. Make both applications, simpleBLAS and simpleFFT by typing make.

3. Run the code (by modifying the submission script from the last section, part A) – the output should show that the error between the library result and the corresponding CPU "Gold" code is very small.

4. Read through the source files to see how the library routines are used, referring to the online documentation for both CUBLAS and CUFFT.

Part C

1. Take the information given in the lecture notes and use it to write a code that uses cuRAND to generate a normal distribution of numbers. ***You may want to use the makefile and submission script from part A***

2. Add a function to your host code that will calculate the mean and standard deviation of the random number distribution that you generate using cuRAND.

3. Write a function that will generate a histogram of the randoms.

4. Use printf() to output your histogram to a file and then using a plotting tool (such as gnuplot) to view the results. You can always copy the results back to your local machine for viewing.

Bonus questions

a. Generate different random number distributions using cuRAND and plot the results.
b. Implement the Box-Muller transform on the CPU and compare the results to those generated by cuRAND (https://en.wikipedia.org/wiki/Box%E2%80%93Muller_transform)

Do not worry if you don't complete all of the above. The aim of this practical is to encourage you to write your own C code and become familiar with some of the common functions.