# Shared_Const_Lab

December 6, 2023

Massive parallel programming on GPUs and applications, by Lokman ABBAS TURKI

## 1 9 Normal cumulative distribution function, using read-only and constant memory allocation, and shared memory

### 1.1 9.1 Objective

The main purpose of this lab is to introduce the student to the use of different memory spaces available on the device. The example presented here is an approximation of the Normal cumulative distribution function that involves the value of some parameters. We consider storing these parameters in a large read-only memory, constant memory, and shared memory. Keep in mind that this is only an exercise in which we show how to use each type of memory space. Indeed, from the beginning of this lab, we already know that the best option for this function is to use registers!

The presented example is the continuation of lab number 5 in which we already studied the use of global and local memory allocation and registers.

As usual, I urge students to open CUDA documentation, especially:

1) the specifications of CUDA API functions within the CUDA_Runtime_API.
2) the examples of how to use the CUDA API functions in CUDA_C_Programming_Guide

### 1.2 9.2 Content

Compile NP.cu using

[ ]:
```
!nvcc NP.cu -o NP
```

Execute NP using (on Microsoft Windows OS ./ is not needed)

[ ]:
```
!./NP
```

As long as you did not include any additional instruction in the file NP.cu, the execution above is not supposed to return any value. At least, no compilation error is detected by the compiler!

In the following questions you will need to include your own code in the file NP.cu, then compile it and execute it. In the following sections, we assume that the device is a pre-Hopper GPU (otherwise add -arch=compute_80 -code=sm_Yy, if Yy>=90).

### 1.2.1 9.2.1 Read-only of parameters in global memory

For kernel NP_GLOreadOnly_k, we want to keep using global memory Glob as in NP_GLO_k but with the read-only data cache load function ___ldg.

a) Since we are not allowed to set the values of Glob within the kernel, do it on the host and send it to the device using cudaMemcpyToSymbol.

b) Complete the syntax of the kernel NP_GLOreadOnly_k.

c) Explain the execution time difference when compared to NP_GLO_k.

### 1.2.2 9.2.2 Read-only of parameters in constant memory

For kernel NP_CST_k, we replace the use of the large read-only global memory in the previous section with a small constant array Cst.

a) Since we are not allowed to set the values of Cst within the kernel, do it on the host and send it to the device using cudaMemcpyToSymbol.

b) Complete the syntax of the kernel NP_CST_k.

c) Compare the execution time of NP_CST_k to NP_GLOreadOnly_k and to NP_GLO_k using !nvprof ./NP.

### 1.2.3 9.2.3 Using shared memory

For kernel NP_SHA_k, instead of making the 7 values constant as with CST, we put the values of these parameters in the shared memory.

a) How should you set the values in the shared memory within the kernel NP_SHA_k?

b) Complete the syntax of the kernel NP_SHA_k.

c) Compare the execution time of NP_SHA_k to NP_CST_k and to NP_GLOreadOnly_k using !nvprof ./NP.

[ ]: