EPFL

Parallel and High Performance Computing

Dr. Pablo Antolín

Series 7

April 10 2025

CUDA - Intro to GPU programming

1 CUDA

These exercises will make you experiment with the main concepts of CUDA programming. To access the code pull the `math454-phpc/exercises-2025` repository.
For all the CUDA exercises we will use the cluster `izar`. The home folders are shared across `hjed` and `izar`, so all your files are accessible from both machines.
For every exercise we provide you some starting code that contains comments marked TODO where you will have to add your code. You will also find a submission script called `script.sh`. In the script all the parameters to submit a slurm job using gpus are there:

- Account: `--account=math-454`
- QOS: `--qos=math-454`
- 1 GPU: `--gres=gpu:1`
- Partition: `--partition=gpu` (the default on `izar`)

To compile you will need to load cuda in addition to a compiler `module load gcc cuda`.

Exercise 1.1: *Hello, World*

This exercise is for illustrative purposes only: In general, it is a bad idea to use a `printf` in a CUDA kernel since it forces useless synchronizations between the host and the device.

- In the folder `hello_world` edit the `hello_world.cu` file to add a `printf` in the CUDA kernel. Try printing the thread and block indexes.
- Be sure to add the proper instruction after calling the kernel to be sure to get the output on the screen.

Exercise 1.2: ***Vector addition***

In this exercise you will experience different implementations of the vector addition kernel. To test before having implemented all the versions of the kernel you can comment the calls to not yet implemented kernels.

- Allocate and deallocate the vectors A, B, and C with CUDA managed memory.
- Implement the vector addition with only 1 thread in `vectorAddOneThread`. Modify the call to the kernel to use 1 thread, 1 block.
- Implement the vector addition with 256 threads and 1 block in `vectorAddOneBlock`. Modify the call to the kernel to use 256 threads and 1 block.
- Implement the vector addition with 256 threads and the proper amount of blocks to have one thread per entry in the array in `vectorAdd`. Modify the call to the kernel to use 256 threads and the proper amount of blocks.

Exercise 1.3: ***Matrix multiplication***

In this exercise we will implement a naïve version of the matrix multiplication

- Look at how the kernel is called in `matrix_mul.cu`.
- Implement the naïve matrix multiplication `in matrix_mul_gpu.cu`