EPFL

Parallel and High Performance Computing

Dr. Pablo Antolín

Solution Series 7

April 10 2025

CUDA - Intro to GPU programming

1 CUDA

Exercise 1.1: *Hello, World*

You have to use `blockIdx.x`, `blockIdx.y`, and `blockIdx.z` for accessing the index of blocks, and `threadIdx.x`, `threadIdx.y`, and `threadIdx.z` for the threads.
After the kernel call, you have to call `cudaDeviceSynchronize`, otherwise the code will end without printing anything.

Exercise 1.2: *Vector addition*

- The first version has to by called with 1 block 1 thread, i.e., `<<<1, 1>>>`.
- The second version is called with `<<<1, 256>>>`. In the kernel, the loop has to go up to `N/blockDim.x`, so the index of the computation is `b * blockDim.x + threadIdx.x`, with `b` the loop counter.
- Finally, the last call is with `<<<N / 256, 256>>>`. In this case, each thread takes care of one entry in the vector, with index `i = blockIdx.x * blockDim.x + threadIdx.x`. If `N` is not a multiple of `blockDim.x`, you have to check than `i` is smaller then `N`, and add an extra block in the call.

Exercise 1.3: *Matrix multiplication*

The kernel is called with

```
dim3 threads = dim3(BLOCK_SIZE, BLOCK_SIZE);
dim3 grid = dim3(device_C.cols() / threads.x, device_C.rows() / threads.y);
```

which means each thread computes one entry in the C matrix. The entry is

```
int j = blockIdx.x * blockDim.x + threadIdx.x;
int i = blockIdx.y * blockDim.y + threadIdx.y;
```

So, you just need to loop over the common dimension between A and B. Here we do not take care of the case where the size of the matrix is not a multiple of the block sizes.