HW_built_Lab

December 6, 2023

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

1 2. Hello World! and built-in variables

1.1 2.1 Objective

The main purpose of this lab is to show the multi-granularity parallelization in CUDA involving blocks, threads, and warps. We show this multi-granularity using an easy example of printing built-in variables. The "printf" function on the device is also studied as well as other GPU specifications that can be obtained with fast "cudaDeviceGetAttribute". When printing built-in variables, feel free to test your own if statement.

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 2.2 Content

Compile HWbuilt.cu using

[ ]:
```
!nvcc HWbuilt.cu -o HW
```

Execute HW using (on Windows machine ./ is not needed)

[4]:
```
!./HW
```

As long as you did not include any additional instruction in the file HWbuilt.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 HWbuilt.cu, then compile it and execute it before answering.

1.2.1 2.2.1 In the main function, use cudaDeviceGetAttribute and then cudaDeviceGetLimit to display

a) The major Y and the minor y of the device compute capability. Explain why this is faster than using cudaGetDeviceProperties.

b) The size of the FIFO buffer of the data transferred to the host when executing printf() on the device. Explain then why should we use cudaDeviceSynchronize on the host when calling printf() on the device.

In the following questions, all compilations will be performed using the compilation options

-arch=compute_60 -code=sm_Yy

, with Y and y from question 1.2.1.a. For example

[ ]:
```
!nvcc HWbuilt.cu -arch=compute_60 -code=sm_75 -o HW
```

1.2.2 2.2.2 Printing Hello World! and built-in variables

a) Fill the kernel print_k with the appropriate code to print "Hello World!" and then call it within the main function.

b) What happens when you call print_k with two blocks and four threads per block?

c) Instead of printing "Hello World!", print the values of threadIdx.x, blockDim.x, blockIdx.x, and gridDim.x.

d) With two blocks and four threads per block, execute the code of question c) multiple times. What do you remark?

1.2.3 2.2.3 More built-in variables printing with 64 blocks and 64 threads per block

a) In print_k, write an if sentence that excludes all threads from printing except threads 0.

b) Execute the code of the question a) multiple times. What do you remark?

c) In print_k, write an if sentence that excludes all blocks from printing except block 0.

d) Execute the code of the question c) multiple times. What do you remark?

e) From questions c) and d), what is the number of threads (lanes or threads of the same warp) that execute in lockstep?

f) What if you answer questions c)d)e) and compile your code with -arch=compute_Yy -code=sm_Yy, with Yy>60? What happens?

1.2.4 2.2.4 Your index variables using built-in variables

a) Defines an index variable to distinguish lanes (threads within the same warp).

b) Defines an index variable to distinguish warps within one block.

c) Defines an index variable to distinguish warps within the whole grid.

d) In what memory space are these variables stored?

e) What happens to indices in a)b)c) when we replace warps with "n" consecutive threads?

[ ]:
```

```