Device_Query_Lab

December 6, 2023

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

1 1. Device query and error handling

1.1 1.1 Objective

The main purpose of this lab is to introduce the student to the CUDA hardware resources and their capabilities. The specification of the GPU will be frequently needed in the following labs. Thus, once the file DevQuery.cu has been configured with the appropriate instructions, it can be compiled and executed whenever students require information about the limitations of the currently utilized device.

This lab session serves as the initial opportunity for students to utilize CUDA documentation, enabling them to discover: 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 1.2 Content

Compile DevQuery.cu using

[ ]:
```
!nvcc DevQuery.cu -o DQ
```

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

[2]:
```
!./DQ
```

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

1.2.1 1.2.1 In the main function, use cudaGetDeviceCount to display the number of available GPUs

Hint: use the CUDA documentation

1.2.2 1.2.2 In the main function, and with device 0, use cudaGetDeviceProperties to display

a) The global memory size. What is the largest single-precision floating-point array that can be allocated on the GPU RAM?

b) The maximum number maxGridSize of blocks that can be launched. What do you notice?

c) The maximum numbers maxThreadsDim and maxThreadsPerBlock of threads per block that can be launched. How do you understand the values given?

d) The size of a warp in terms of threads.

e) The shared memory size per block.

f) The number of registers per block.

g) The size of 2D textures. Would it be possible to have it entirely in the cache?

h) The number of CUDA cores. (hint: each multiprocessor contains usually 128 CUDA cores)

1.2.3 1.2.3 Analyze the results from Section 1.2.2

a) How long should be the sequences of diverged execution of warps? (hint: search for "diverge" keyword in CUDA documentation)

b) How do you justify that the number of threads should be (and not must be) a power of 2?

c) What can limit a program from launching the maximum number of blocks?

d) What can limit a program from launching the maximum number of threads?

e) What can trivially make us find a compromise between the number of blocks and the number of threads to be used?

1.2.4 1.2.4 Recall cudaGetDeviceProperties on the device 20

a) What if you change the 0 argument in cudaGetDeviceProperties by 20?

b) What kind of cudaError_t do you get?

c) Use now testCUDA and see what happens.