Introduction: CUDA

CUDA is a hardware design and framework that allows the programming of the GPU nuclei and resource management.

CUDA Architecture: Compute Unified Device Architecture)

- Architecture introduced in 2006.
- Allows the scientific community the access to the GPU resources.
- Brings support for high level programming languages, such as C/C++.
- Unifies the use of independent processors, which come from the classic architecture.
- Uses an Heterogeneous Model: this is formed with two hardware elements, Device and Host.

The Hardware (GPC) design has the following:

- SM Unit (all green squares and blues, etc), also called MultiProcessors. Each multiProcessor has a max limit of blocks that can be processed in this SM. These have:
  - SP Unit (each green square), also called CUDA cores or nucleus. Search for this in CUDA website.
- A cluster (GPC) is the group of SM's or Streaming Multiprocessors.
- L2 Cache Shared Memory

Figure 1: Image

Heterogeneous Model

- **Host**: CPU. Less cores or nuclei.
- **Device**: GPU or Graphics Card.

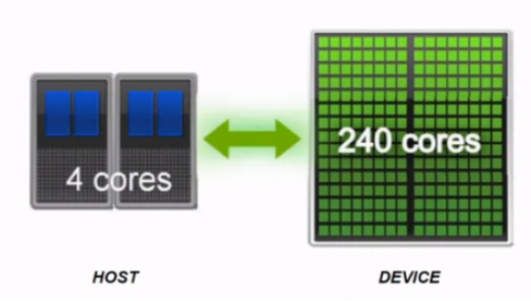

Figure 2: Image

Processing Stream

Starts with the Host (Sequential) and goes then to Device (Parallel) and then Host, etc...

Kernel, Threads, blocks and Grids

- **Kernel**: gives the instructions to all the cores or organizes the cores. The code snippet that you want to execute in parallel.
- **Blocks**: cores are organized or grouped in blocks. The yellow squares. A block groups threads.
- **Grid**: a group of one or more blocks (Green big square). Each GPU has only one Grid.
- **Warp**: a group of 32 threads, they're inside blocks as blocks have threads. A warp **is** physically executed in parallel.
- A single thread is executed in a single CUDA core. Commonly: Thread = CUDA core.

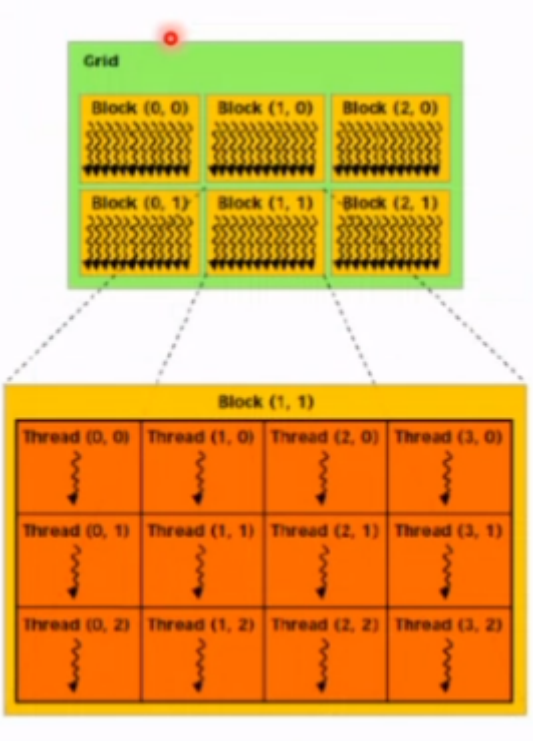

Figure 3: Image

Not every time everything runs in parallel, the first warp goes (32 threads per block) first, and then the next warp and so on. When a block is executed, not the whole block is executed, just its first 32 threads (warp), then other 32, etc.

- A GPU is a group of multiprocessors.
- A block has threads, but you can have different amounts of threads in many blocks, just careful not to exceed `threadsInBlock x Blocks` <= `maxThreadsPerMultiProcessor`.
- Grids and blocks are three dimensional.

Perspectives

- Hardware level: many cluster (multiprocessor)
- Software level: one 3D grid with blocks.

Lab 01

Print the main properties of your machine's GPU.

Solution

```
#include "cuda_runtime.h"
#include "device_launch_parameters.h"

#include <stdio.h>


int main()
{
    int device = 0; // to store the number of devices we have
    int* count = &device;
    cudaGetDeviceCount(count); // needs a pointer to store the
        result
    // a device is a gpu card
    printf("Device count: %d\n", device);

    cudaDeviceProp properties;
    cudaDeviceProp* pProperties = &properties;
    cudaGetDeviceProperties(pProperties, device - 1); //
       device int is an index, we have one so index is zero
    printf("Name: %s\n", properties.name); // name of the
       device
    printf("multiProcessorCount: %d\n", properties.
       multiProcessorCount);
    printf("maxBlocksPerMultiProcessor: %d\n", properties.
       maxBlocksPerMultiProcessor);
    // the sum of all the threads in each block
    printf("maxThreadsPerMultiProcessor: %d\n", properties.
       maxThreadsPerMultiProcessor);
    // max number of threads per block
    printf("maxThreadsPerBlock: %d\n", properties.
       maxThreadsPerBlock);

    // Grids dimensions
    printf("maxGridSize x axis: %d\n", properties.maxGridSize
       [0]); // max limit of blocks in x axis in the grid
    printf("maxGridSize y axis: %d\n", properties.maxGridSize
       [1]);
    printf("maxGridSize z axis: %d\n", properties.maxGridSize
       [2]);

    // Block dimensions (tweak but until the multip is <=
       1024)
    printf("maxThreadsDim x axis: %d\n", properties.
       maxThreadsDim[0]); // max limit of threads per
       dimension in block
    printf("maxThreadsDim y axis: %d\n", properties.
       maxThreadsDim[1]);
    printf("maxThreadsDim z axis: %d\n", properties.
       maxThreadsDim[2]);
```

```
35
36      return 0;
37  }
```

Output

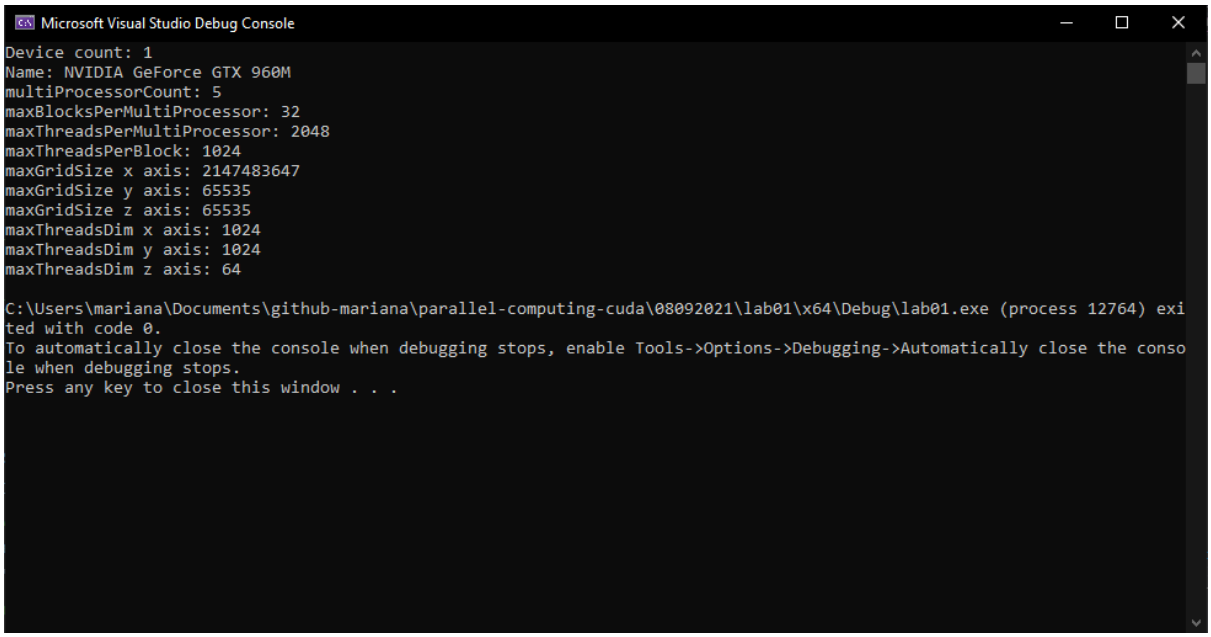

Figure 4: Image