

# PROGRAMMING PROJECTS LABORATORY

OpenCL

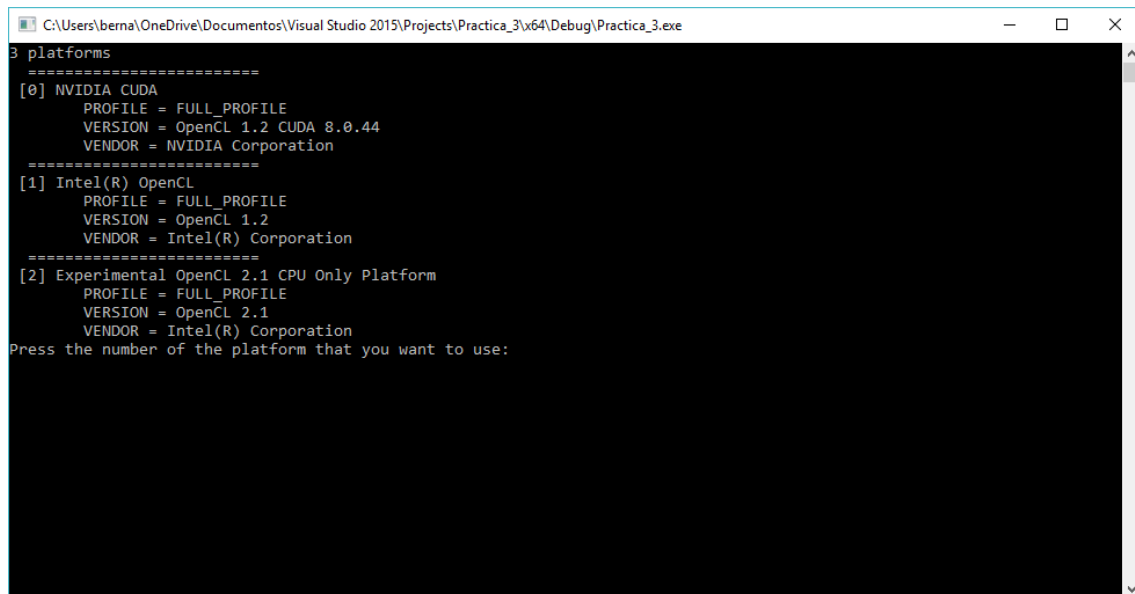


Bernat Galmés Rubert  
2016-2017

User manual:

Select platform:

When you execute the program, you will see the next window:

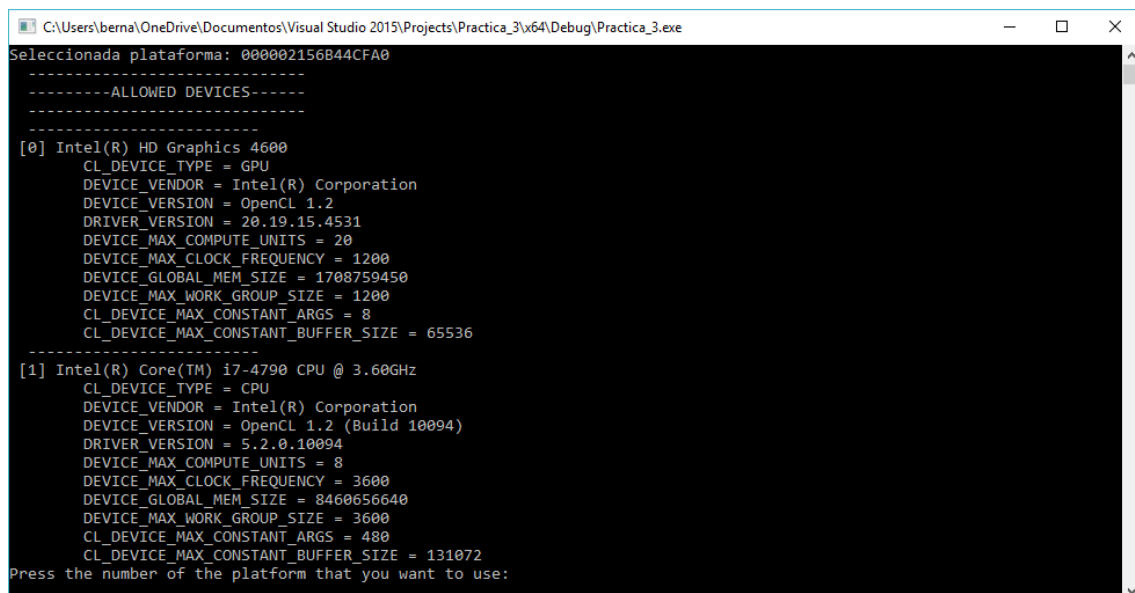


```
C:\Users\berna\OneDrive\Documentos\Visual Studio 2015\Projects\Practica_3\x64\Debug\Practica_3.exe
3 platforms
=====
[0] NVIDIA CUDA
    PROFILE = FULL_PROFILE
    VERSION = OpenCL 1.2 CUDA 8.0.44
    VENDOR = NVIDIA Corporation
=====
[1] Intel(R) OpenCL
    PROFILE = FULL_PROFILE
    VERSION = OpenCL 1.2
    VENDOR = Intel(R) Corporation
=====
[2] Experimental OpenCL 2.1 CPU Only Platform
    PROFILE = FULL_PROFILE
    VERSION = OpenCL 2.1
    VENDOR = Intel(R) Corporation
Press the number of the platform that you want to use:
```

For select a platform, must insert the number of the platform that you want to use and press enter key.

Select device:

After have selected a platform, you must select the device that you want to use to execute the program. The next screen allows you to do that:

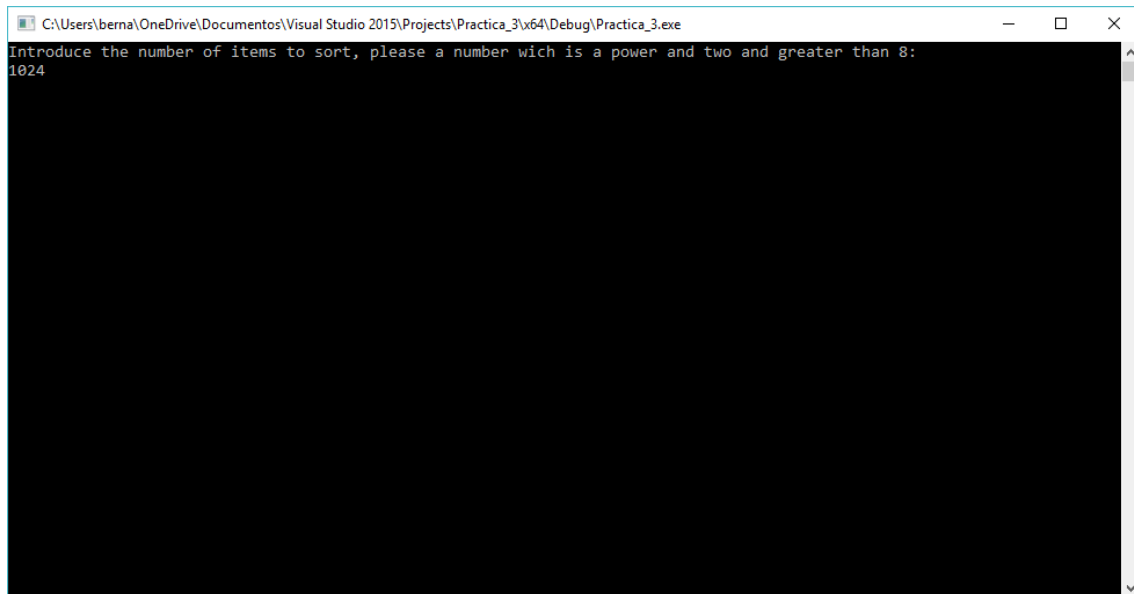


```
C:\Users\berna\OneDrive\Documentos\Visual Studio 2015\Projects\Practica_3\x64\Debug\Practica_3.exe
Seleccionada plataforma: 000002156B44CFA0
-----ALLOWED DEVICES-----
-----
[0] Intel(R) HD Graphics 4600
    CL_DEVICE_TYPE = GPU
    DEVICE_VENDOR = Intel(R) Corporation
    DEVICE_VERSION = OpenCL 1.2
    DRIVER_VERSION = 20.19.15.4531
    DEVICE_MAX_COMPUTE_UNITS = 20
    DEVICE_MAX_CLOCK_FREQUENCY = 1200
    DEVICE_GLOBAL_MEM_SIZE = 1708759450
    DEVICE_MAX_WORK_GROUP_SIZE = 1200
    CL_DEVICE_MAX_CONSTANT_ARGS = 8
    CL_DEVICE_MAX_CONSTANT_BUFFER_SIZE = 65536
-----
[1] Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz
    CL_DEVICE_TYPE = CPU
    DEVICE_VENDOR = Intel(R) Corporation
    DEVICE_VERSION = OpenCL 1.2 (Build 10094)
    DRIVER_VERSION = 5.2.0.10094
    DEVICE_MAX_COMPUTE_UNITS = 8
    DEVICE_MAX_CLOCK_FREQUENCY = 3600
    DEVICE_GLOBAL_MEM_SIZE = 8460656640
    DEVICE_MAX_WORK_GROUP_SIZE = 3600
    CL_DEVICE_MAX_CONSTANT_ARGS = 480
    CL_DEVICE_MAX_CONSTANT_BUFFER_SIZE = 131072
Press the number of the platform that you want to use:
```

Here, you must insert the number of the device that you want to use and press enter.

### Select array size:

When you have selected the platform and the device, the OpenCL are ready to run. Now, you only need to insert the number of elements that you want to sort.



In this screen you must insert the number of items that you want that you algorithm sort. The number must comply the next constrains:

- The number must be a power of two ( $2^n$ )
- The number must be greater than 8.

After have inserted the number of items, the program will generate an array with the number of items introduced, and will fill it with random values. Then the sort algorithm runs with this array.

### Results:

When the algorithm have finished to run you will see the next window with the result of the execution:

```
C:\Users\berna\OneDrive\Documentos\Visual Studio 2015\Projects\Practica_3\x64\Debug\Practica_3.exe
Original array:
42, 18468, 6335, 26501, 19170, 15725, 11479, 29359, 26963, 24465, 5706, 28146, 23282, 16828, 9962,
492,
Sorted array:
42, 492, 5706, 6335, 9962, 11479, 15725, 16828, 18468, 19170, 23282, 24465, 26501, 26963, 28146,
29359,
Time sending data: 16 ms
Time processing data: 16 ms
Time retrieving data: 0 ms
Total time alghoritm: 32 ms
Time all process: 125 ms
```

The console now is showing the next information:

- First, The content of the array before it has been sorted.
- Second, the array sorted by openCL.
- Third, the spended time by the program.

The sections of the times are splitted in 5 parts:

- Time sending data: Time that the program has used to send data from the host memory to the device memory.
- Time processing data: Time spended executing kernel functions.
- Time retrieving data: Time spended getting the data from the device memory.
- Total time alghoritm: Time spended executing the sorting algorithm.
- Time all process: Time carried out executing all the sorting process.

\*Note: time retrieving data is 0 because the program only read from the device memory one time to read the results.

## Conclusions:

The most important aspect that I have to solve, is that, you can only sincronize the work-items inside an item-group. The alghorithm that I have before was a direct iterative translation of the explanation in the Wikipedia:

[https://en.wikipedia.org/wiki/Bitonic\\_sorter](https://en.wikipedia.org/wiki/Bitonic_sorter)

Where each array position was a work-item. And I throw all the kernels together, and control the memory with barriers. The problem was that I have to had all the work-items in the same work-group, and there is a limit of the numbers of work-groups that may be.

To solve that I change the algorithm, where I have one work-item for each comparison, and I swap the memory in the same kernel.

The new algorithm works like the next image, each arrow is a comparison(one kernel), the big boxes are the stages of the algorithm, and all the red boxes in the same column are a phase of the stage. The program control the stages in the host, and throw the work-items for each phase of a stage.

