Programming projects laboratory

OpenCL

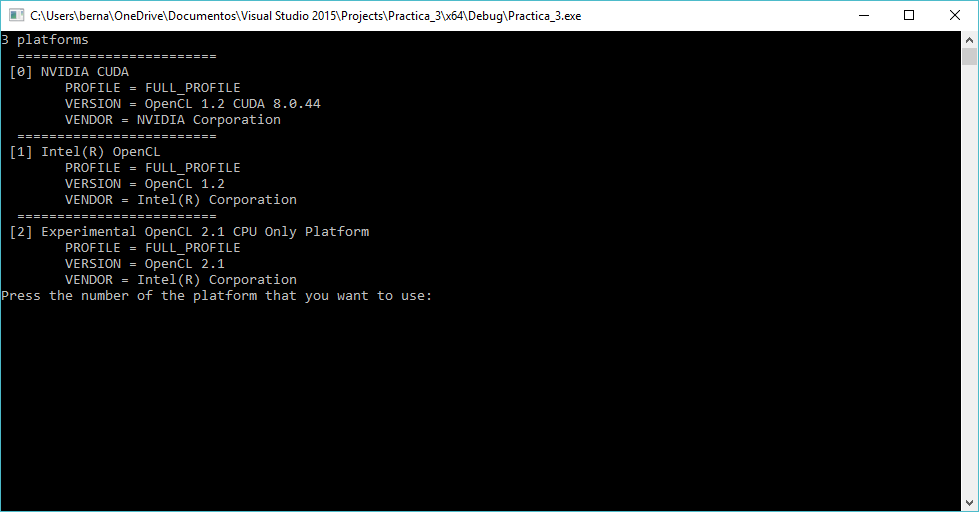
Bernat Galmés Rubert

2016-2017

# User manual:

## Select platform:

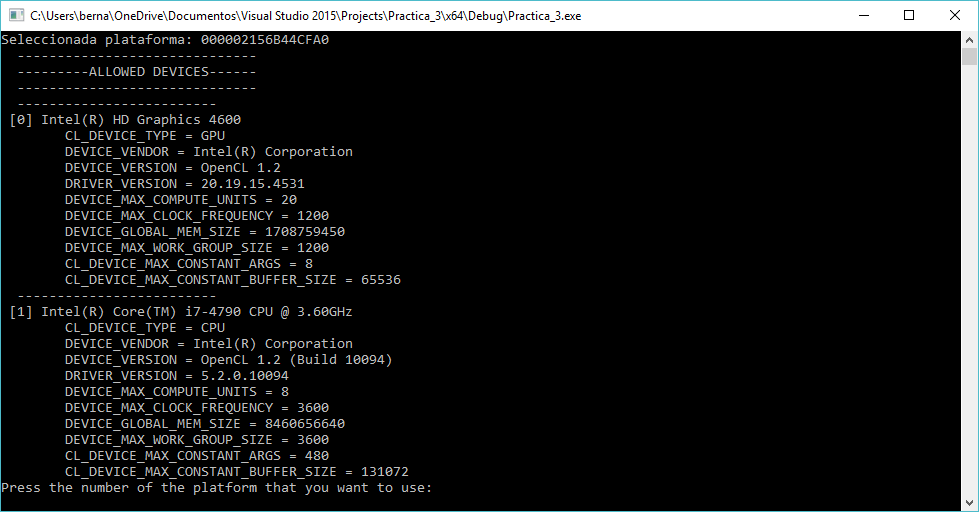
When you execute the program, you will see the next window:



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

## Select device:

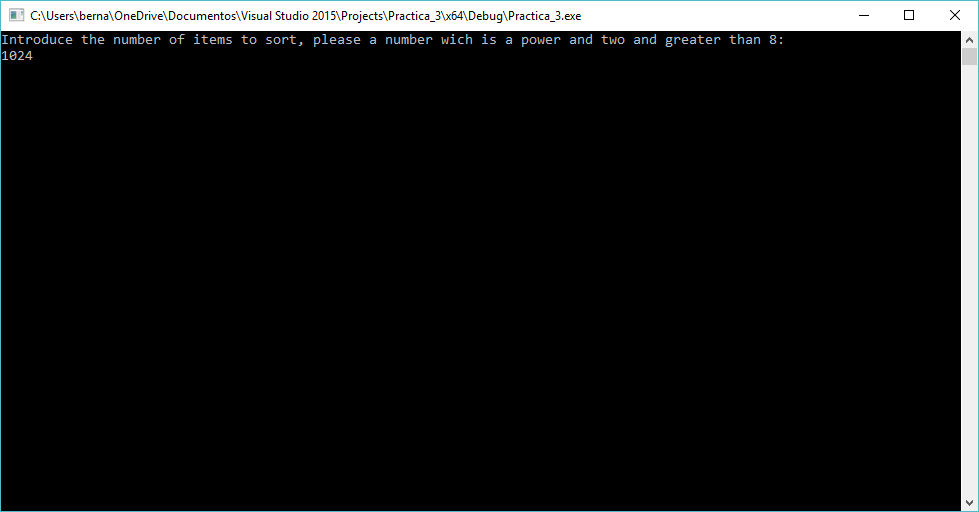
After have selected a platform, you must select the device that you want to use to execute the program. The next screen allows you to do that:



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.



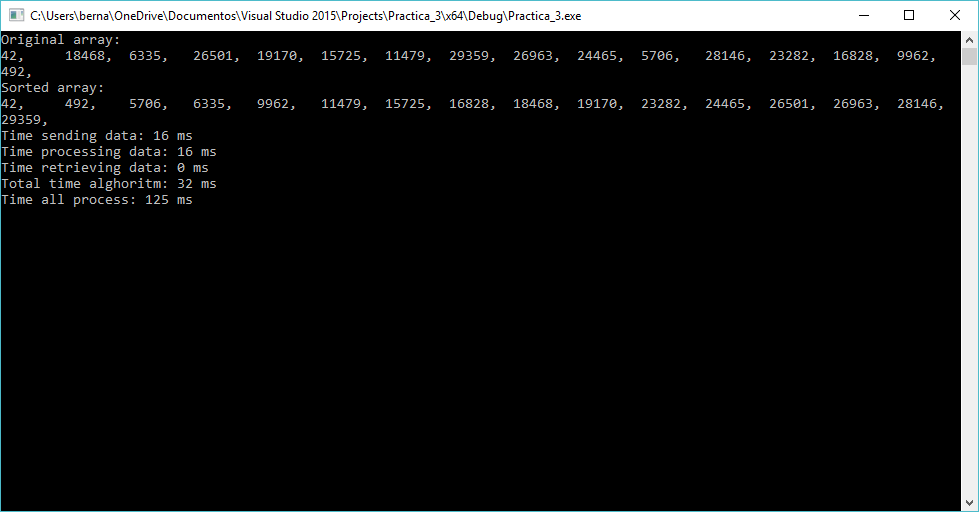
In this screen you must insert the number of items that you want that you algorithm sort. The number must comply the next constrains:

* 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:



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 alghorithm: Time spended executing the sorting alghorithm.
* 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>

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 comparation, and I swap the memory in the same kernel.

The new alghorithm works like the next image, each arrow is a comparation(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.

