AGH	Module: **Introduction to CUDA and OpenCL**	Report title: **Report 6**		
	Instructor: **dr hab. inż. Szumlak Tomasz**			
	Name and surname: **Marcin Filipek** **Anca Dărăbuț**	Group: **11**	Date: **02.01.2020**	Grade:

1. overview

Another framework is represented by OpenCL. Just as CUDA, it deals with massive parallelism, but its use is more universal. Consequently, the device we are working with can be any CPU or GPU. It comes up with a degree of automatisation, as the performance is diminished.

For OpenCL, the context plays an important role. It is the central piece, a special object that needs to be aware of everything, i.e. data, queues, kernel.

The structure of the program written in OpenCL is generally the same as in CUDA. We create the data and kernel, send the data to execution then copy it back. OpenCL has some characteristics regarding the way data is handled. Flags are used to determine whether the data is used for being read or written on the host or device. Synchronization is not done explicitly, but relying on queues and contexts. The shape of the grid is defined by the range, restricting the user control over the environment. Last, but not least, the kernel is injected as a string.

In this laboratory, we studied the behaviour of two different applications. The former one is the classical vector addition, in order to compare the performance of OpenCL with CUDA, while the latter one is consecutive addition.

2. timing profiles for OpenCL

As always, we conducted some tests to check the performance. It was harder than in CUDA, because of big differences between the same input data.

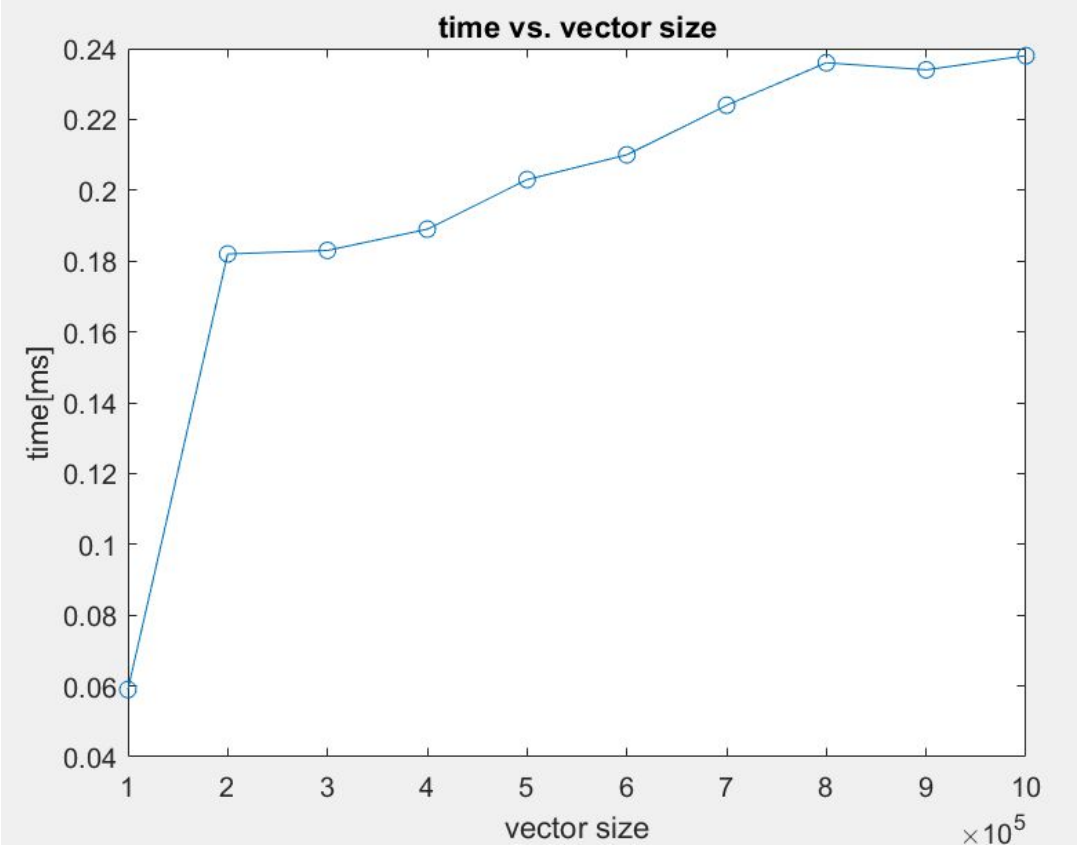
2.1. vector add

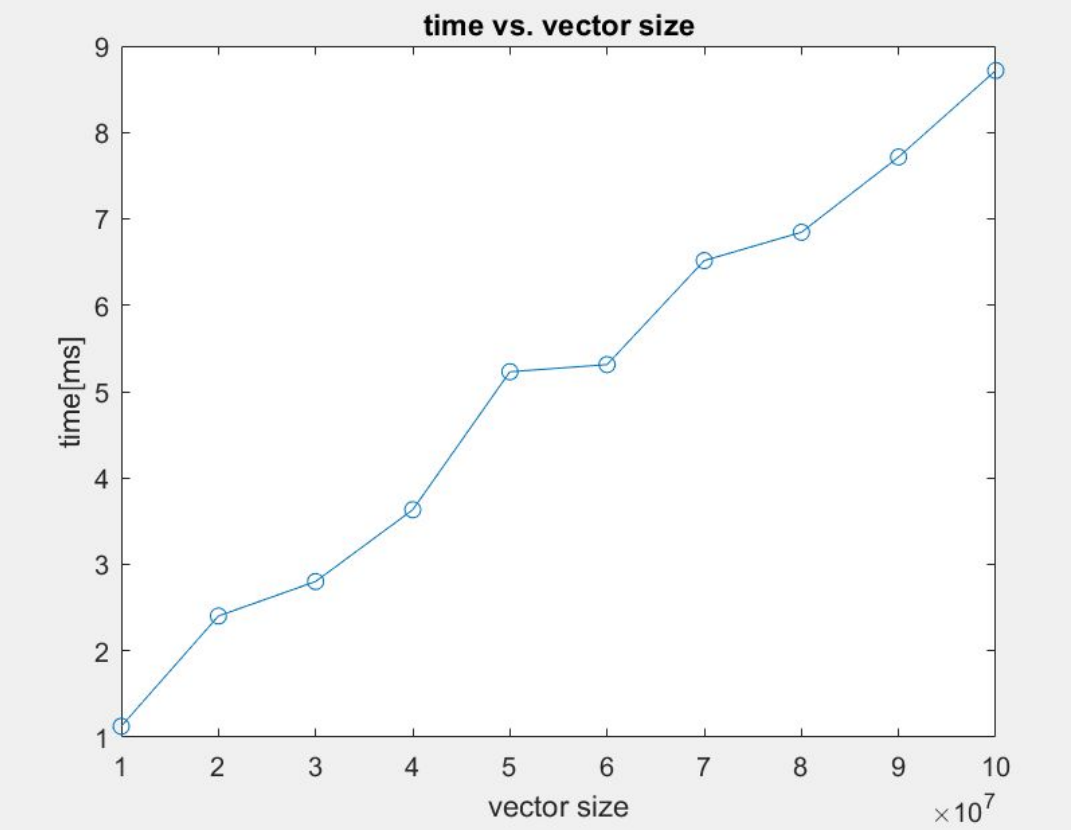
The first application is the basic vector addition.

For instance, time measurements for the 100 million elements long vectors, we have obtained the results we have listed below:

time (ms): 10.184 8.502 8.324 9.453 11.096 8.717

It is probably caused by the fact that we did not define the “grid” manually. The program might have distributed the resources differently in every execution. Then we decided to run the test several times for every size and pick the lowest result. The results are presented in the plots below:



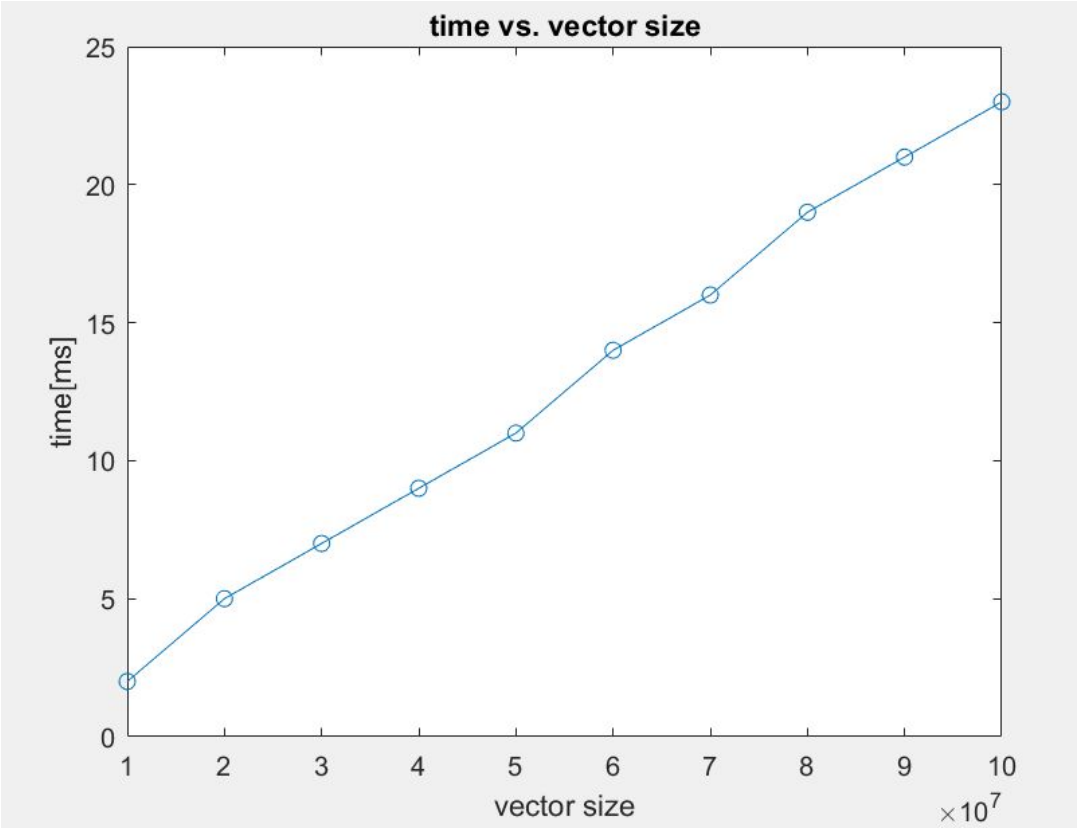


The time has an almost linearly increasing tendency, thus, for bigger data sizes, the time is greater. However, in the first case, when the number of elements is smaller with a few orders of ten, the behaviour is peculiar. The jump between 100000 and 200000 is quite significant (the time is increased three times).

2.2. consecutive addition

The second application is the consecutive addition.

In this case, 4 different vectors are added together. The complexity of the program stands into the kernel execution and data handling. The kernel of the vector addition remains untouched, thus it is supposed to be executed three times. Consequently, the vectors have different missions. The ones created by the user are read_only, the intermediary ones read_write and the result is write_only. The functions of the data are described through OpenCL data flags.



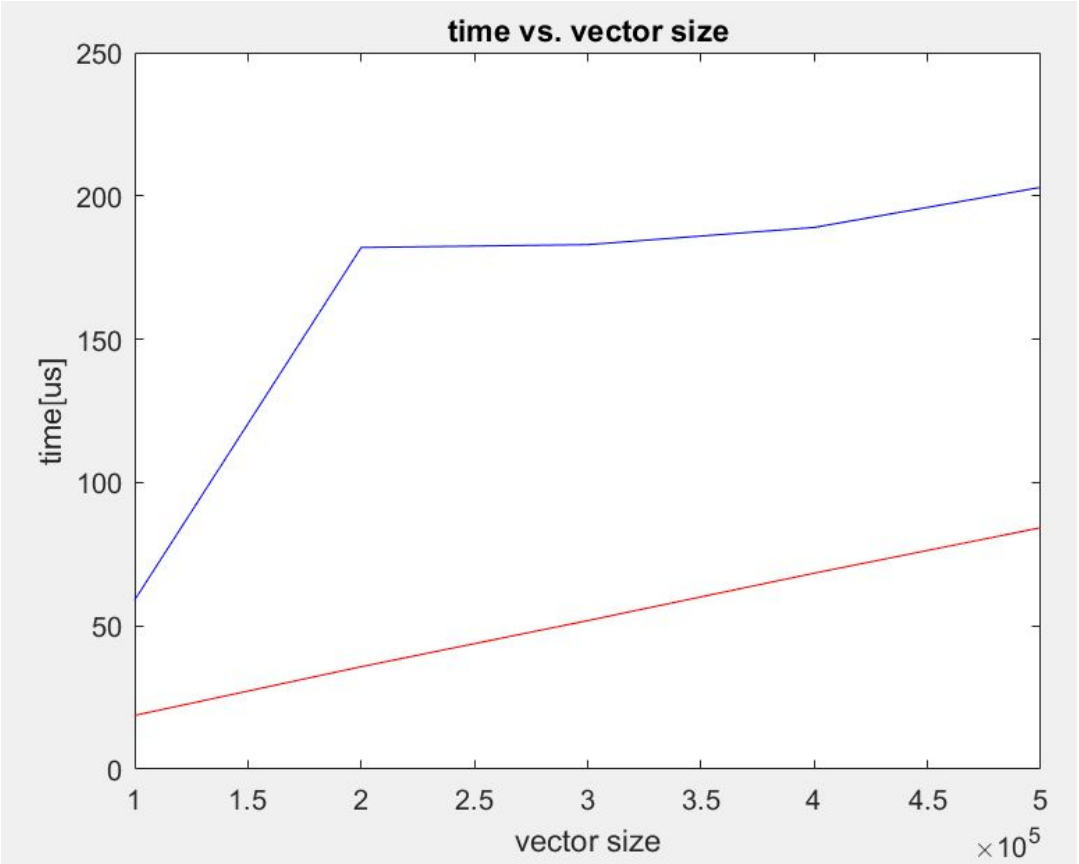
As expected, the time profiling is similar with the one of the basic vector addition, i.e. it is almost linear, but the values of the times are almost 3 times greater in the case of consecutive addition, because, in this case, the kernel is called and executed 3 times.

3. comparison between CUDA and OpenCL time performance

After the tests, the question appeared: is it faster or slower than CUDA? Our experience tells us that it should be slower, due to some automatisation. In CUDA we have everything under control, so we are able to optimise the device for certain program, but if we change the device, we have to do it again. Also, CUDA works only with Nvidia GPUs, in contrast to OpenCL, where the device and be any processing unit which has multiple cores.

vector size	CUDA time [us]	OpenCL time [us]
100 000	18.75	59
200 000	35.75	182
300 000	51.84	183
400 000	68.39	189

500 000	84.26	203



The graph presented above shows the comparison between the two performances. CUDA time profiling is presented with red colour, whereas OpenCL times are in blue.

The conclusion that can be clearly drawn from these time profilings is that CUDA is faster than OpenCL even with the basic implementation (the basic vector addition performed in the first laboratory classes).