AGH University of Science and Technology

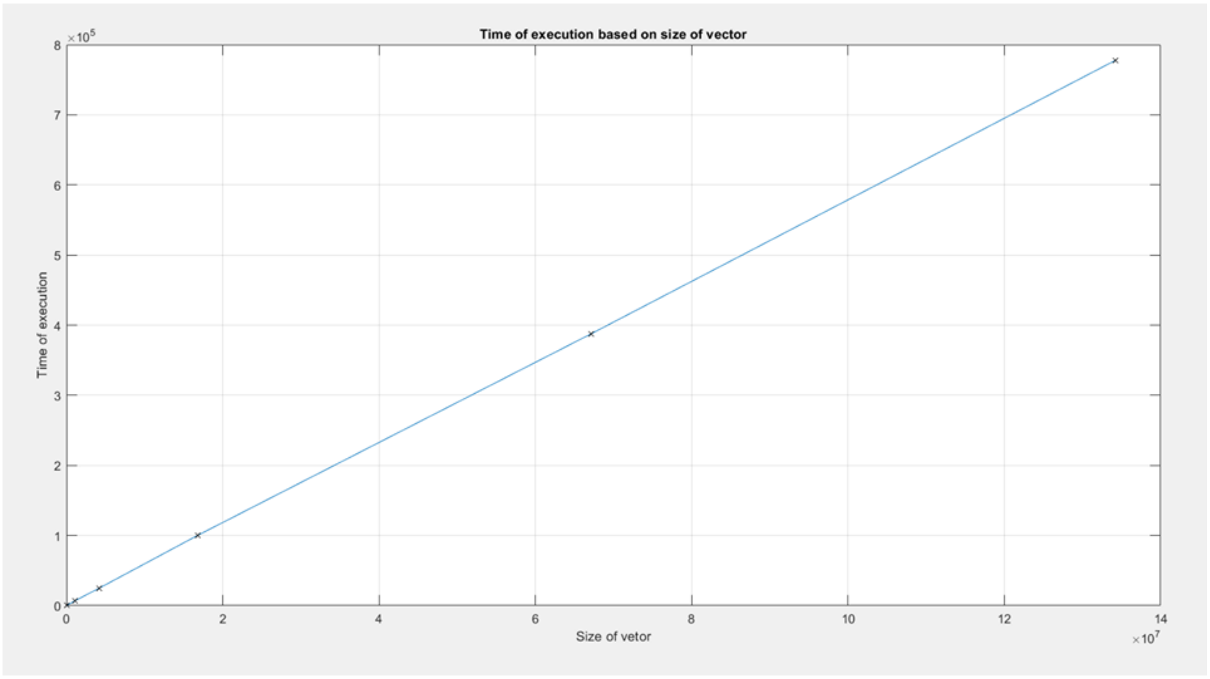
Introduction to CUDA and Open CL

Lab 2

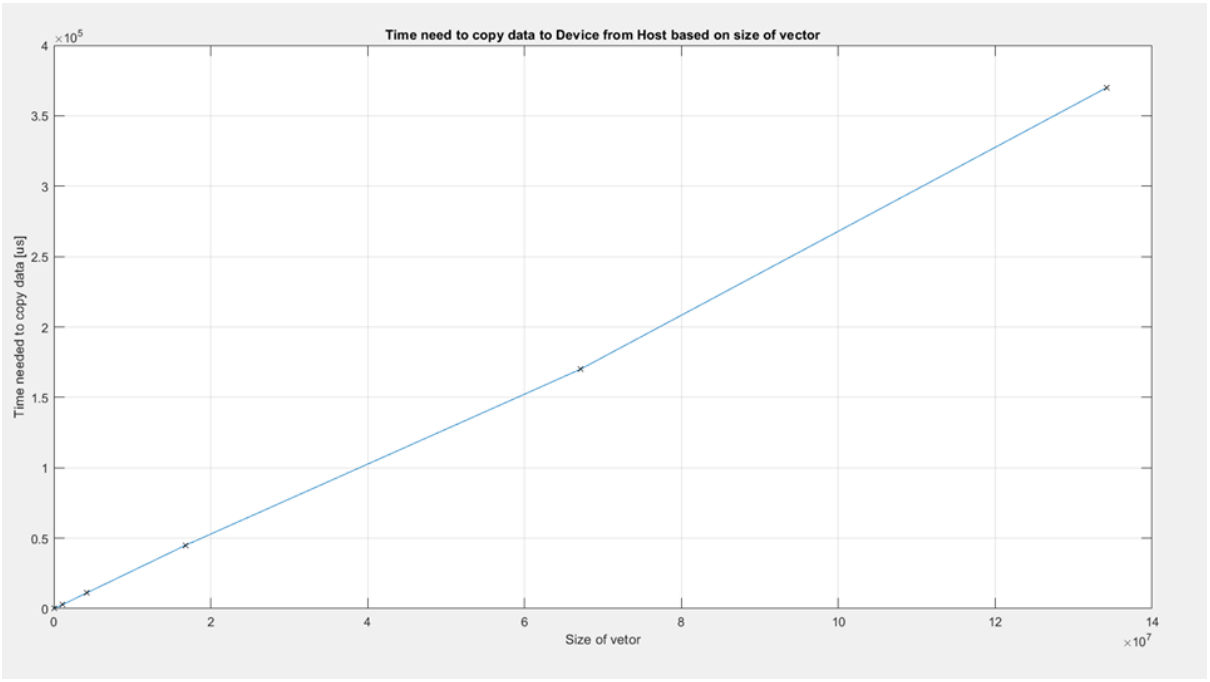
Michał Kunkel
Wiktor Żychowicz

1. How size of data we proceed impact our performance.

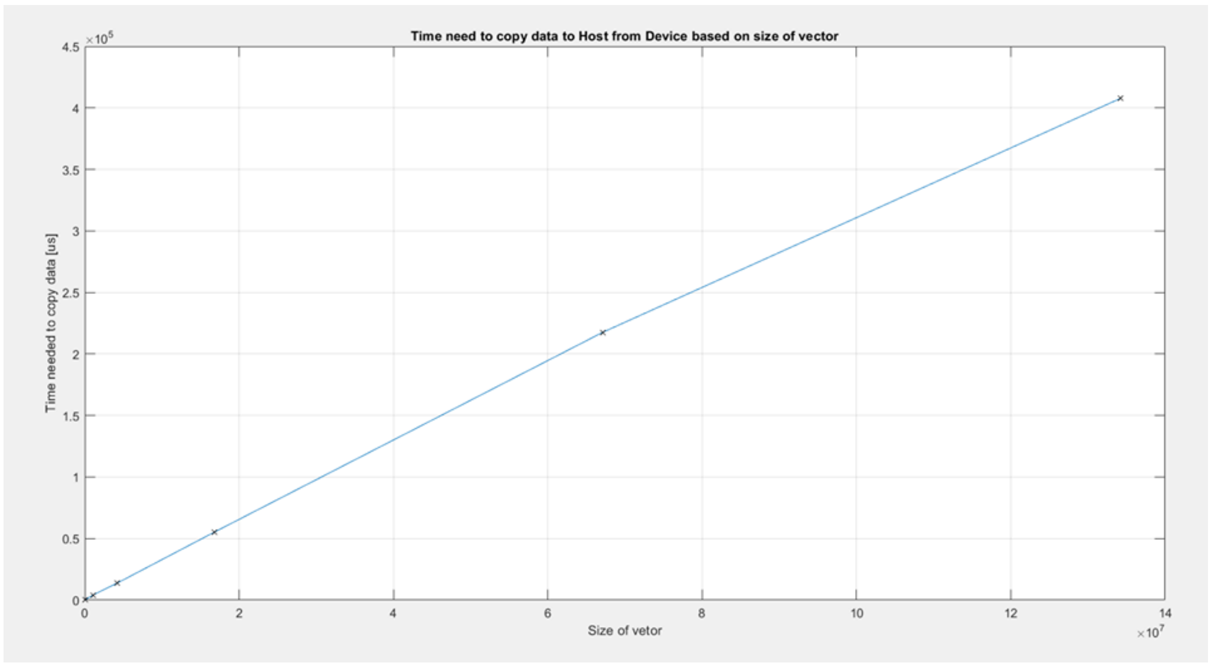
In computation with GPU size meter. In terms of how many GPUs we have and consequently how many of single resources (size of available memory(L1,L2,L3, GPU HDRR5,), number of SP(streaming multiprocessors) and CUDA cores). Moreover size of data we procced and it relation with resources is a crucial matter. We tested this relation using nvprof profiler and here are results:



Plot 1: Summary time of execution in relation with data size.



Plot 2: Time consumed by fetching data to device.



Plot 3: Time consumed by fetching result to host.

All test was made on NVIDIA 1060GTX. Kernel was adding 2 vectors consist of floats. Then we fetch result back to the host.

As we can see on all plots in typical case relation is linear. Some deviation can be noticed when data set is small and “setting up” of GPU is a significant variable. Plot 1 shows us that computation (in this particular example, when computations are independent which one another and do not need synchronization with host) does not affect general trend.

2. When things goes wrong.

So far we have linear relation between size of the data and time we need to send it. First anomaly occur when vector can not fit into GPU memory and CUDA scheduler must fetch it in more than one cycle. Execution time goes significantly up because of practical multiplying of memory related processes. What can be even more disturbing for program flow is necessity of synchronization host and device each time we fetch data.

When size is even bigger(like 2 to the power of 28) whole program stop to works as expected. We know problem is hardware related because overflow of int(type in which we store vector size) occur in 2 to the power of 30 scenario(in our test environment). However we failed to establish what is cause of such an error we are sure that it is resource related.

3. Different ways of memory management.

There exist more than one way to manage memory when using CUDA. We can do it “manually” via special CUDA functions but we can also atomized process and sacrifice some of the control by using cudaMallocMenager. No matter which one we choose physically all data must be fetch the same way. Malloc Manager however hides single functions and take care of them. This allows to type ang think less however not always is optimal and can lead to some misconceptions(provoke programmers to type somewhat ridiculous code detached from real way of how memory travel form host to device) about how data is fetch in reality. In the end in both ways we do the same thing but manager is easier to use however less transparent.