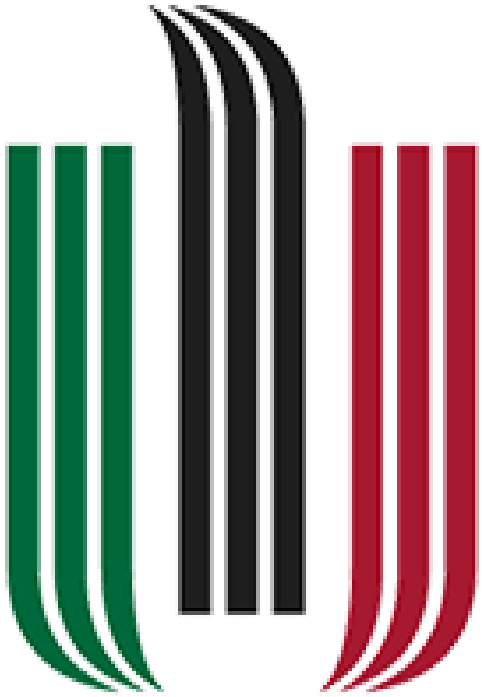

AGH

AGH University of Science and Technology

Introduction to CUDA and Open CL

Lab 1

Michał Kunkel
Wiktor Żychowicz

1. Gaining data about hardware

One of the biggest advantages of CUDA is wide variety of compatible devices which are highly scalable. The same program(with some minor tweaks) can be run efficiently on laptop GPU and on professional workstation containing clusters of high end's devices. We can research online specification of our configuration or use some useful C extensions provided by NVIDIA to check it locally. On laboratory classes we used premade program named "deviceQuery" which does research instead of us and provides good amount of information.

2. Testing performance

CUDA programming to some extended can be optimized by using theoretical knowledge but GPUs are complicated devices and we can not be sure of our performance until me check it in real applications. So we need to experiment with grid layouts, block compositions and various ways of data fetching to find sweet spot for specific program. Fortunately NVIDIA provides us some useful tools to do such a thing. We can use very powerful(a lot of options, highly editable output) terminal profiler(nvprof) or it's graphical equivalent in NVIDIA Nsight IDE. We generate ours plots using programs mentioned above.

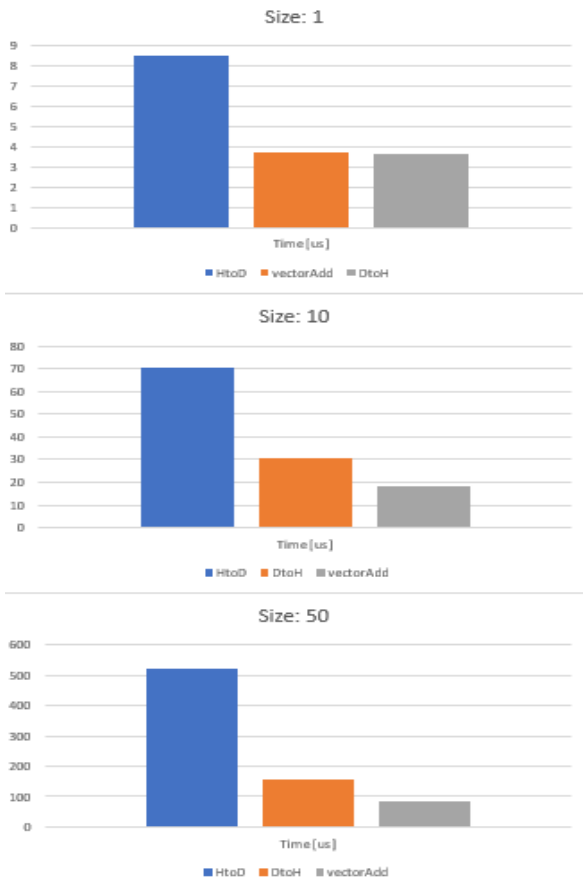

Figure1: Time complexity of individual program parts according to vector size(in 10^{th} of thousands of int).

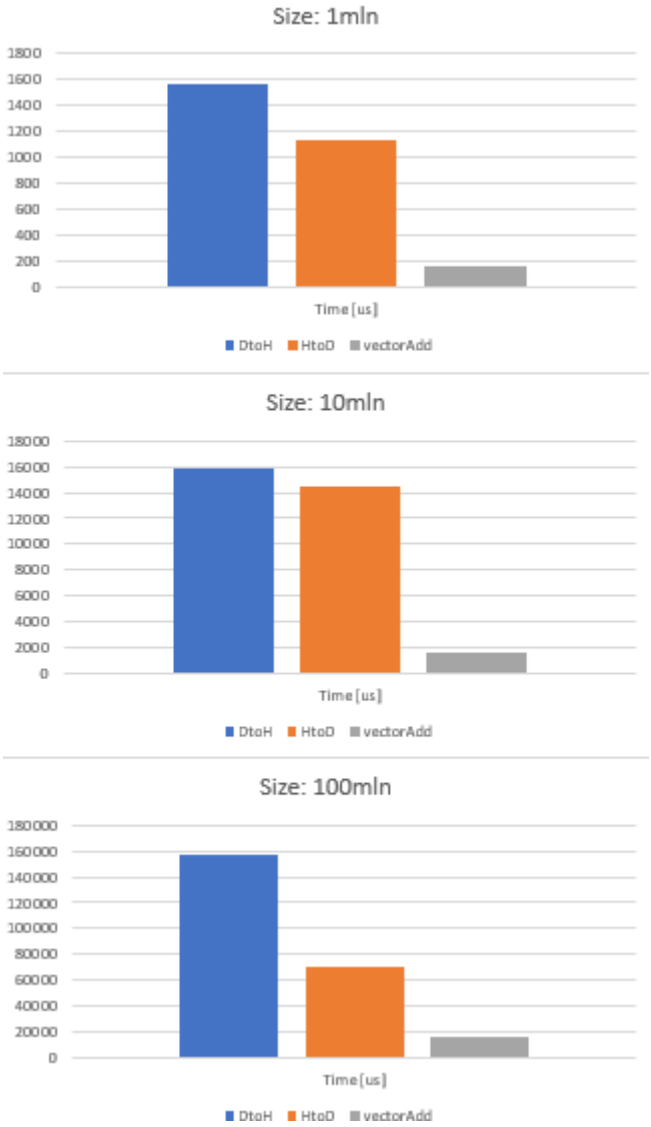

Figure2: Time complexity of individual program parts according to vector size(integers)

On figure1 we can see that computation times is less and less significant as we increase data size. By looking on figure 1 and 2 at the same time we can see that time consumption on coping memory from host to devices is significantly bigger when we operate on huge data sets. During our test we crushed program applying vector of 1mld elements. We can say we found upper limit of capacity for our card.

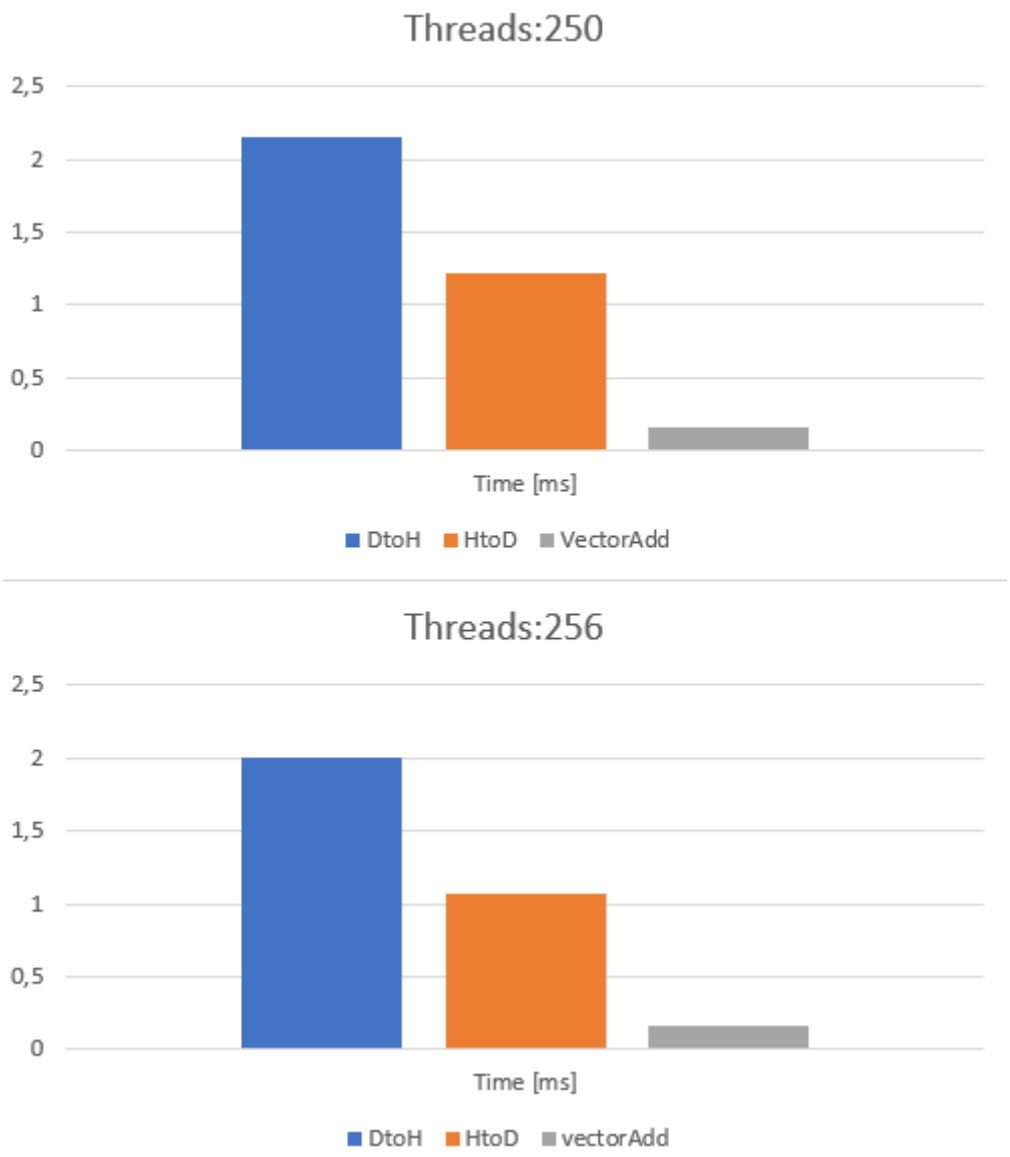

Figure3: Performance according to number of threads in block

As we can see on figure3 threads which are eligible for division by 32 perform better than other number of threads. We called it “magical number”.

3. Data management.

One of the biggest CUDA restrictions is the way we set our data flow. There are a lot of kinds of memory. We have multiple levels of caches on CPU(host) and GPU(device) and some other storage spaces like RAM or hard drive. What is important we have not got access to all our storage on all our devices. Some of them are shared (like hard drive) ant others are exclusive. The main division exist between CPU memory and GPU one. Theirs is also a rule that “closer” storage

source is significantly faster(orders of magnitude). Process of fetching memory between host and device is highly costly in time and require synchronization of botch(in fact it demands other hardware as well like PCI Express) what have an enormous impact on efficiency. That is why, when we program in CUDA me must pay attention to memory management. We should keep our data on the GPU as long as possible. However we cannot write complicated kernels(functions executing on GPU) because design of CUDA cores keep them very simply. The way around is to create pipeline workflow of different instructions executing on the same data(like assembly line production).

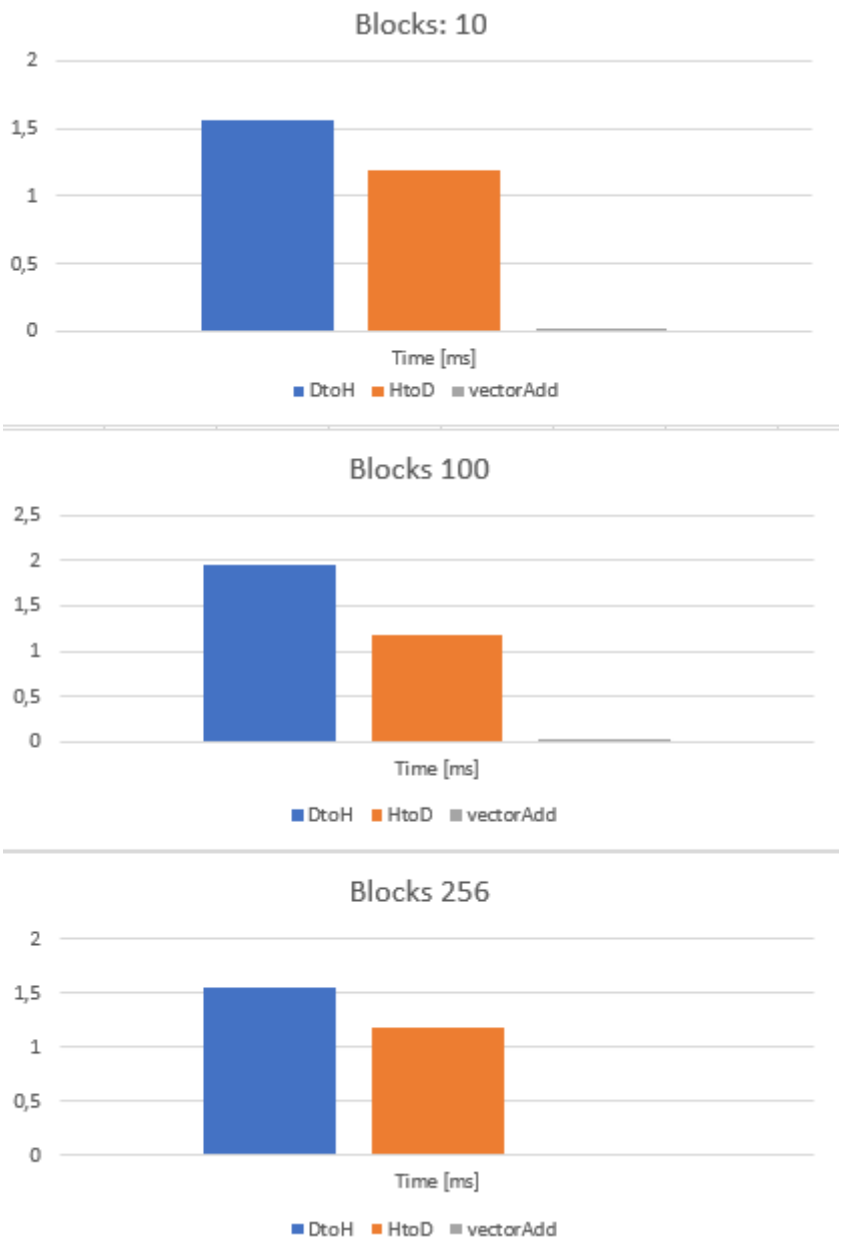

Figure4: Impact of blocks number on execution time.

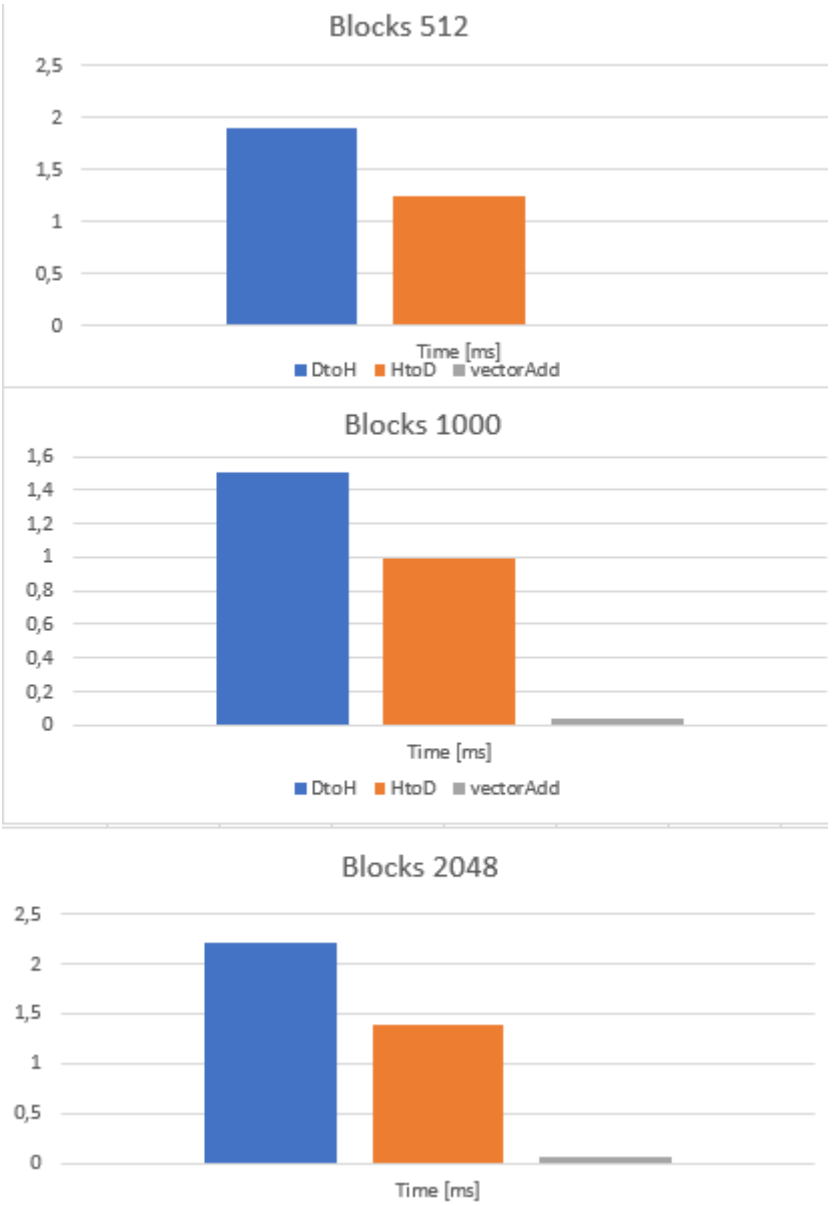

Figure5: Impact of blocks number on execution time(continuation).

4. Calculation time.

Ideally we hope for as long calculation time as possible. In reality usually the bigger part of timeline is taken by memory allocation and other memory-related tasks. Once data lands on GPU and GPU is set(drivers etc.) we start to see advantages of parallel programming. Gain(in time or in lower power bill) which we can obtain hugely depends on our task's level of parallelism. Hopefully there is a lot of it in our surrounding(loops and many other). GPU prefers simple tasks like addition eventually multiplications and simple types like integers or floats. Deciding to using elaborate math functions like sinus, square root or double precision numbers (double type) significantly increase time require to finish task.

5. Summary

CUDA offers occasion to write portable code which can be optimize in a close to hardware way. We can gain very much thanks to wide range of application of parallel programming but we must be aware of the great enemy of highly scalable environments which is data management. We can maximize our performance by selecting good programming model and by using some smart tricks.