Report 2

Introduction to CUDA and OpenCL

Basic strategies of accelerating C/C++ applications.

Dariusz Biela

Karol Sawicki

1. Introduction

At the second laboratory classes we wanted to do some code improvements to obtain more performance in our CUDA accelerated applications. To do that we prepared kernel codes for matrix addition, Hadamard product, which is very similar to the addition code, and the Dyadic product. We wanted to obtain our optimization with more advanced processing grid preparations, that could made our code more parallel. We were working only with square matrixes.

2. Calculations and measurements

To do our tasks we prepared functions for allocation, data initialization, verification, that can be done by CPU and our kernels. For each kernel we decided to do measurements with 2D2D Processing Grid and 1D1D Processing Grid, because changing the grid layout is the simplest way to obtain better performance. We decided to do our measurements the same way, that we did this a week before – we prepared loop, that each time tries to allocate 100 times bigger matrixes, than the run before. This is some kind of a logarithmic progression, because we were increasing our matrixes 10 times in each dimension.

For each matrix size we measured time for 10 kernel runs and calculated the uncertainty. Our calculations were performed with following formula:

$$u = \sqrt{\Sigma(x_i - \bar{x})^2}$$

In our formula n stands for number of measurements, that means 10. We decided to show our uncertainties in the following table, the column name stands for matrix dimension size.

Table 1 Uncertainty for each operation with given matrix size.

	10	100	1000	10000
Add2D	0,023470923	0,004161981	0,029954	6,355805
Add1D	0,007491616	0,004161981	0,029954	6,355805
Hadamard2D	0,007250644	0,004161981	0,029954	6,355805
Hadamard1D	0,006907292	0,004161981	0,029954	6,355805
Dydadic2D	0,007057096	0,004161981	0,029954	6,355805
Dydadic1D	0,006578054	0,004161981	0,029954	6,355805

We prepared graphs that comparing time for kernel execution with 2D2D grid and 1D1D grid.

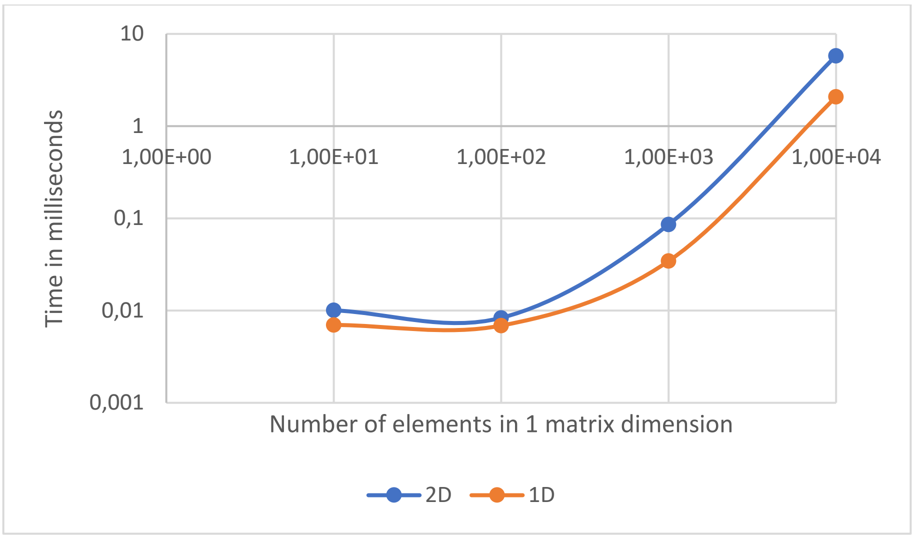

Figure 1 Matrix Addition

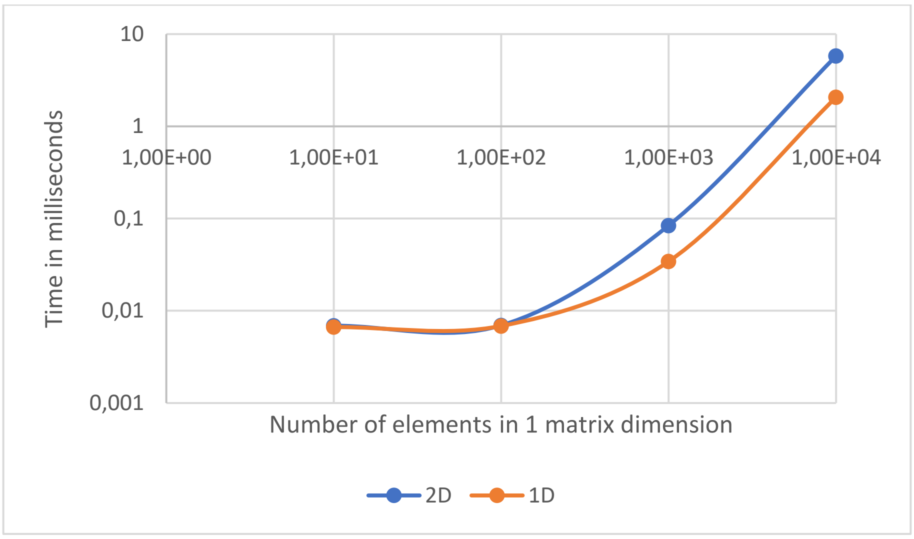

Figure 2 Hadamard Product

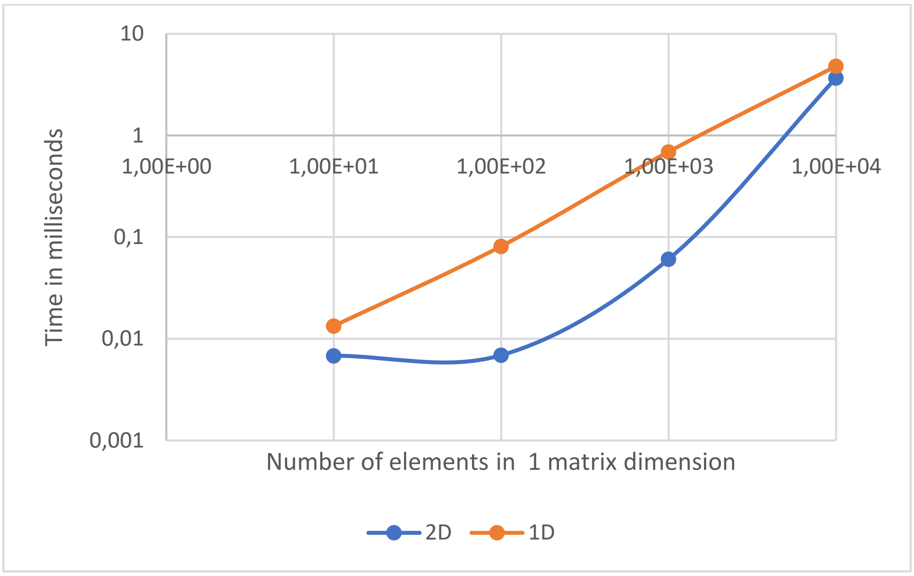

Figure 3 Dyadic Product

At the end we decided to compare kernel execution time for int ant float data types. As matrix addition and Hadamard product are very similar, we choose to compare only one of them.

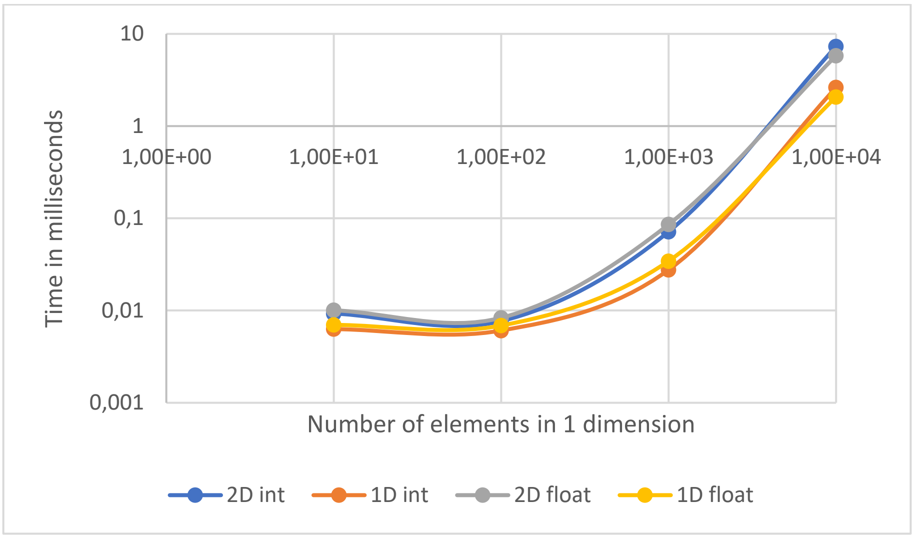

Figure 4 int and float matrix addtion comparison

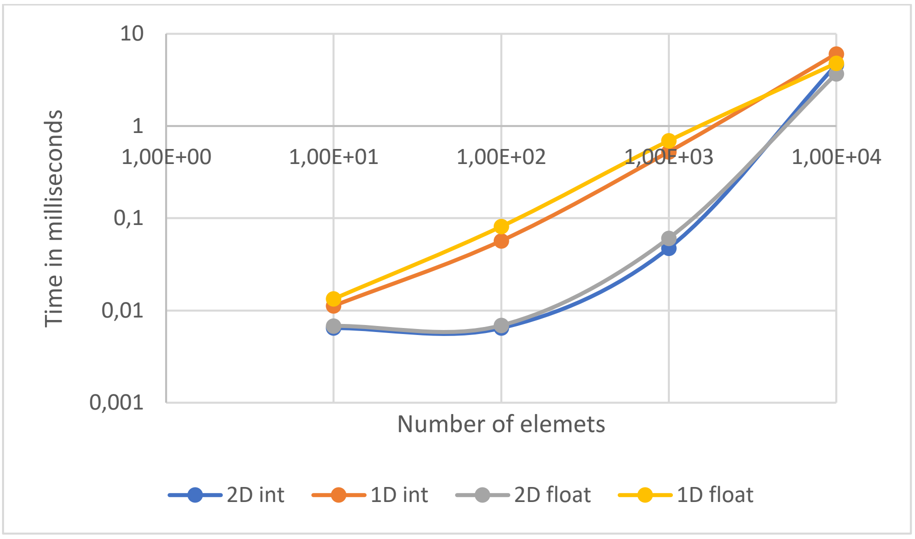

Figure 5 int and float dyadic product comparison

3. Conclusions

Processing Grid preparations are really simple and we could improve kernel execution time only by changing the layout of it. This is not very complicated, but can benefit a lot, we should try different layouts to check which one we should use.

Kernels for different grid layouts should not be the same in every case, like for Dyadic product.

Uncertainty can increase, when we increasing data portions, this could be caused by the less consistent memory access times.

We can observe that there is no the best grid layout, for some cases better is 1D1D, for the other better can be 2D2D grid. We always should prepare code for testing and measurements, before we decide which one we should use. This could take some time, but it will be totally worth, when we have to prepare massive calculations with so much data.

Time for kernel execution with different data types may vary, but for int and float is really similar, this is because the size of each element of this types is the same. In that case we should remember that not only the execution time matters, but the memory too – if we are working with double precision numbers our execution time will be longer, but our data will be much smaller.