Introduction to CUDA and OpenCL

Lab 3

Kacper Kapuściak

Introduction

In the third laboratories, our main task was to master the thread coordination and understand how the layout of the processing grid could improve the parallelization of our code. We've learned it by implementing and measuring the execution of algorithms both on integer and floating-point numbers.
These algorithms were:

1. Matrix addition
2. Hadamard product
3. Dyadic product

The measurements were performed for both on 1D:1D (that is the 1D grid and 1D blocks) and 2D:2D grid layouts.

All operations were executed on Nvidia RTX 2060 graphics card.

Measurements

For each algorithm, I have made 10 measurements for both integer and float numbers and took an average of the execution time using. I have settled for using just 4 points i.e. matrices 10x10, 100x100, 1000x1000, and 10000x10000. The results are presented below.

Matrix addition algorithm

I've started with an A + B matrix addition on integer numbers. In this example 2D:2D layout was about 37% faster than 1D:1D grid layout. It is like this because of a huge difference in time execution on 10000x10000 matrices. The difference was 175 milliseconds.

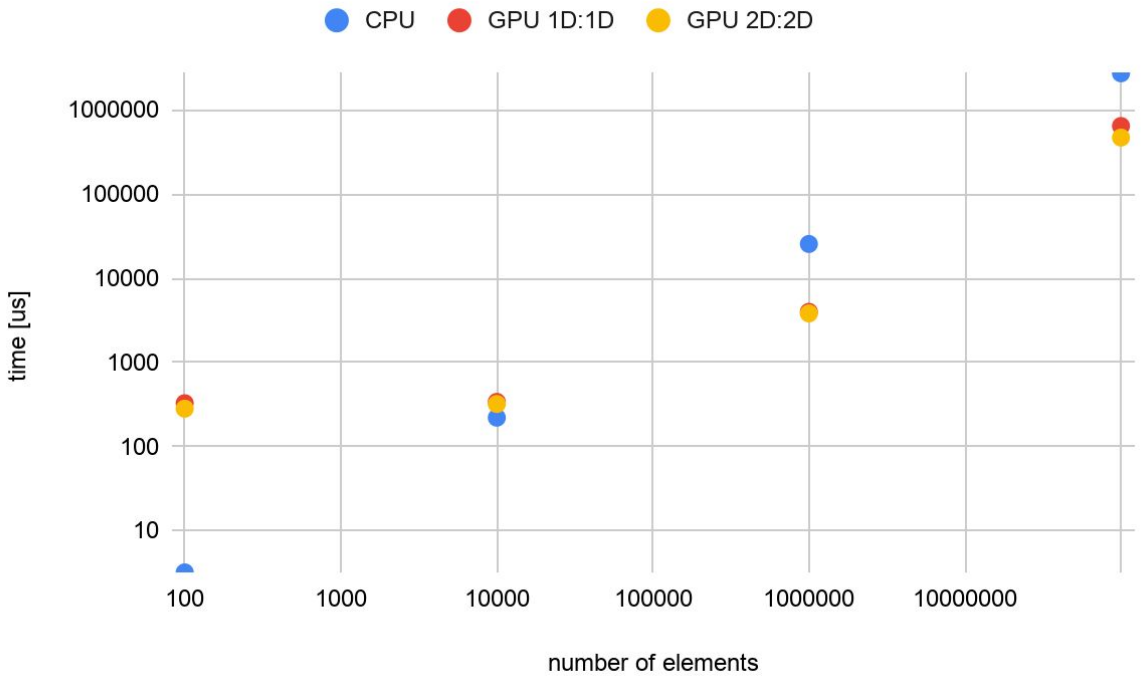

Chart 1. Matrix addition of integer numbers on CPU, GPU with 1D:1D and 2D:2D grid layout

Next, I've made the same measurements for the same algorithm but on floating-point numbers as can be seen on Chart 2. In this case difference between 1D:1D layout and 2D:2D was very small. Smaller than 1%. But on average the calculations on floating-point numbers at least for this algorithm were faster on floating-point.

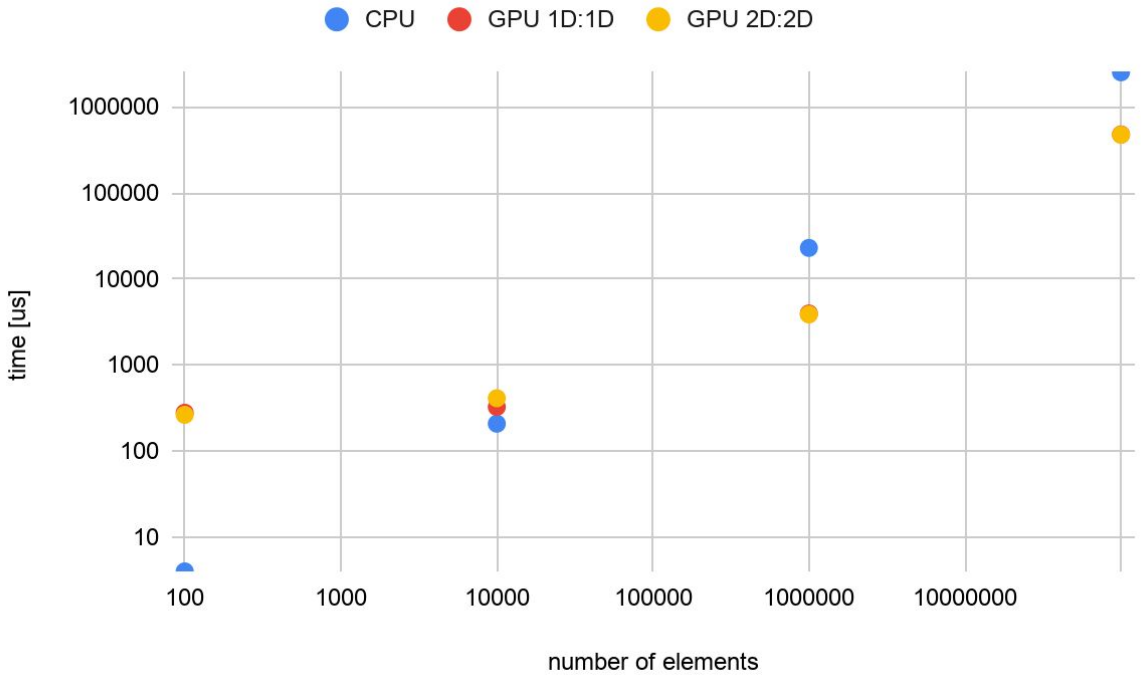

Chart 2. Matrix addition of floating-point numbers on CPU, GPU with 1D:1D and 2D:2D grid layout

Hadamard product algorithm

Next, I've made a similar experiment for the Hadamard product algorithm. I've made measurements for both integer and floating-point numbers in matrices. This time the execution time differs for 1D:1D layout and 2D:2D layout slightly in terms of what is faster for bigger and what is faster for smaller matrices. But on average both of the techniques benchmark identically (the difference is within 1%).

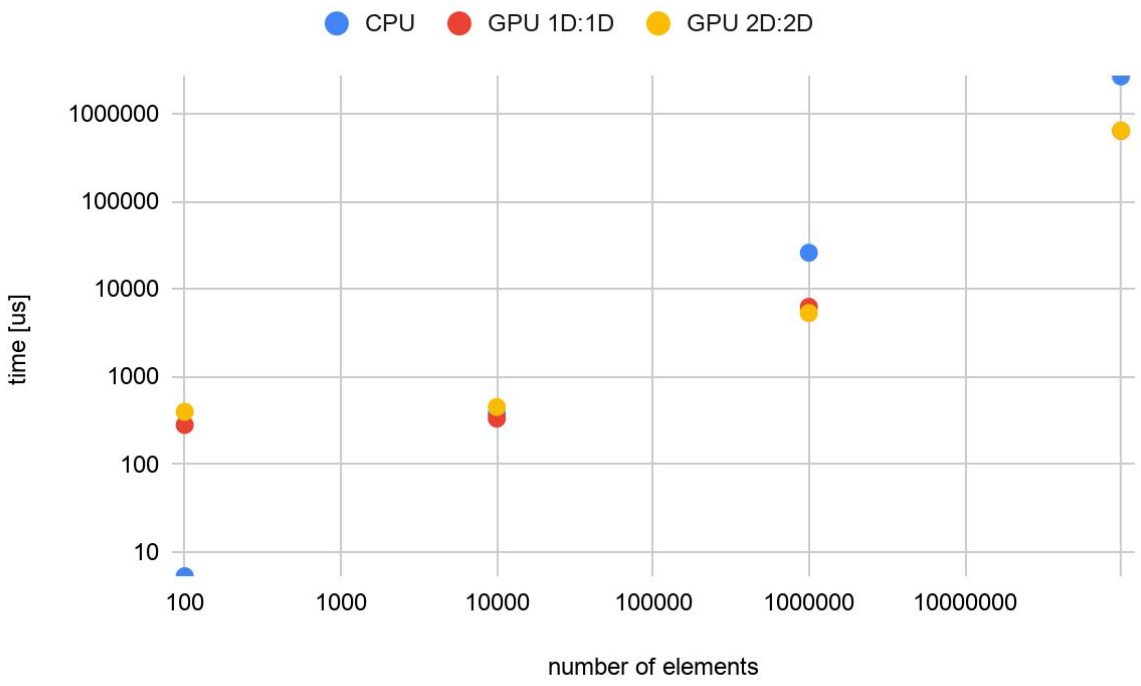

Chart 3. Hamarad product of integer numbers on CPU, GPU with 1D:1D and 2D:2D grid layout

For floating-point calculations of the Hamarad product, we can see a considerable performance boost for 2D:2D layout grid. For every matrix size, 2D grid was faster and on average the performance boost was around 26%. As per usual, compared to CPU calculations performed on GPU are faster for matrices bigger than around 100x100.

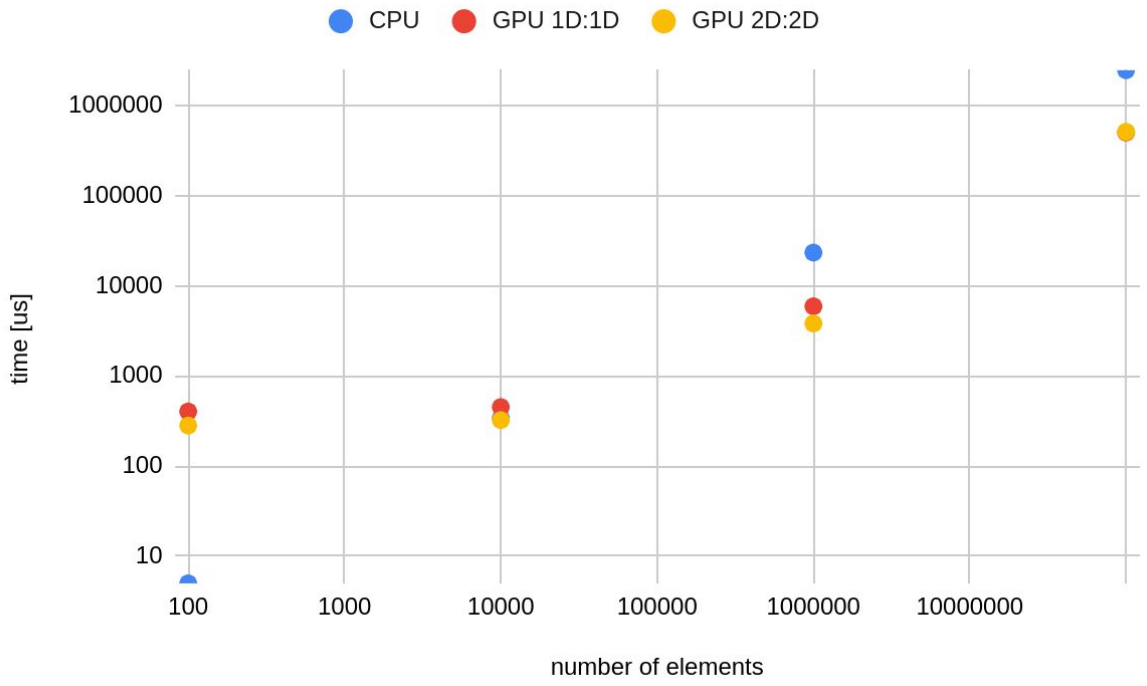

Chart 4. Hamarad product of floating-point numbers on CPU, GPU with 1D:1D and 2D:2D grid layout

Conclusion

For simple calculations and on a small set of numbers making calculations on GPU doesn't have any advantages because fetching data to and back GPU takes longer than a calculation take on CPU. However, for bigger matrices, starting from around 100x100 calculation on GPU is faster. From this point, we shall consider the optimizations of the grid layout. In most cases 2D:2D layout was faster or had no impact on execution time.