Introduction to CUDA and OpenCL: Reduction

Ilona Tomkowicz, Zofia Pieńkowska

February 16, 2020

Contents

1 Exercise goal

The goal of this exercise was to implement the code performing parallel reduction of computations. Some algorithms, like calculating a sum or finding minimal or maximal value in a set, can be performed faster if reduction is implemented. The data is divided into parts on which the algorythm is performed, and its results form another set on which the algorythm is performed, and so on, until only an assumed number of elements is reached (e.g. one in case of adding elements).

2 Implementation and collected results

In this exercise, the sum of all elements in a vector was calculated. To measure the time needed for the computations, `<time.h>` library was used. Also, basic CPU computations were performed to check whether the results of GPU computations were correct. Table 1 presents times needed to add n elements. Time elapsed as a function of n is plotted on fig. 1.

Number of elements n	**Time [ms]**
10000	0
100000	10
500000	20
1000000	90
2000000	280
3000000	540
5000000	1460

Table 1: Time needed to add n values stored in a vector, using parallel reduction.

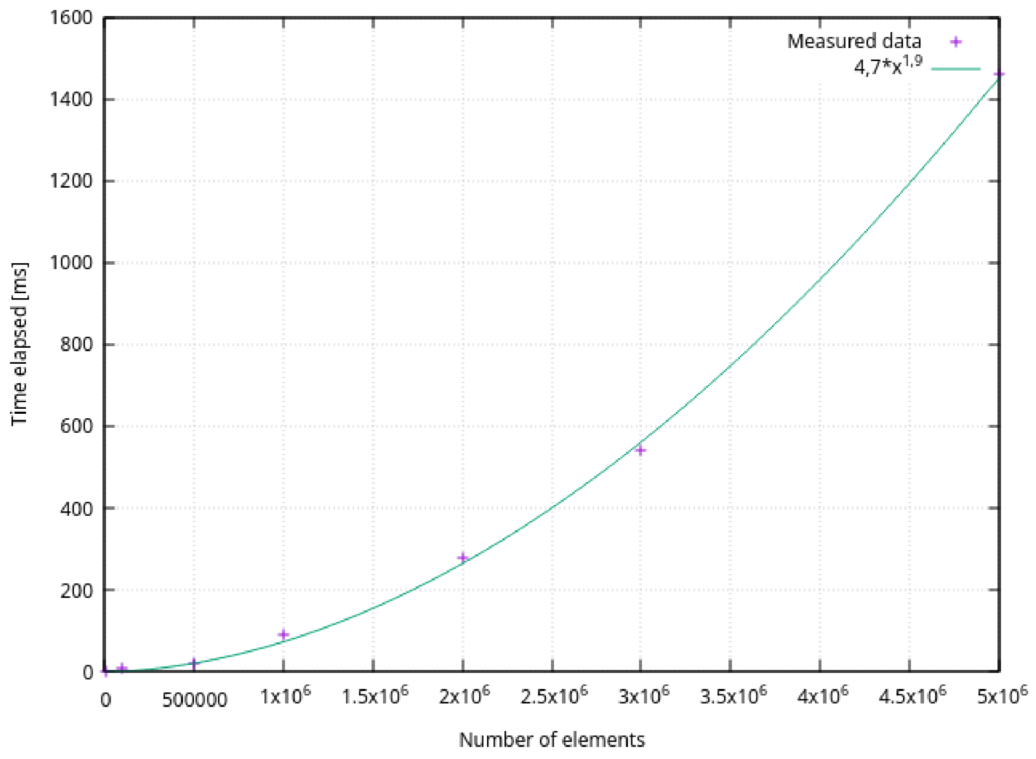

Figure 1: Time elapsed as a function of number of elements. Exponential function has been fitted to measured data, with parameters a=4,7 and b=1,9.

3 Conclusion

Reduction code has been implemented successfully and allows for the computations to be performed faster than while using a naive algorithm. Some inequalities of sums computed by CPU and GPU have been observed, which have increased with the number of elements summed. It has been thus concluded that these inequalities result solely from inaccurate storing of floating point values, and not from a faulty algorithm implementation. Time needed for computations is growing exponentially with the number of elements.