Introduction to CUDA and OpenCL
Lab 9 report

7.05.2020
Piotr Litwin
Paweł Skalny

Introduction

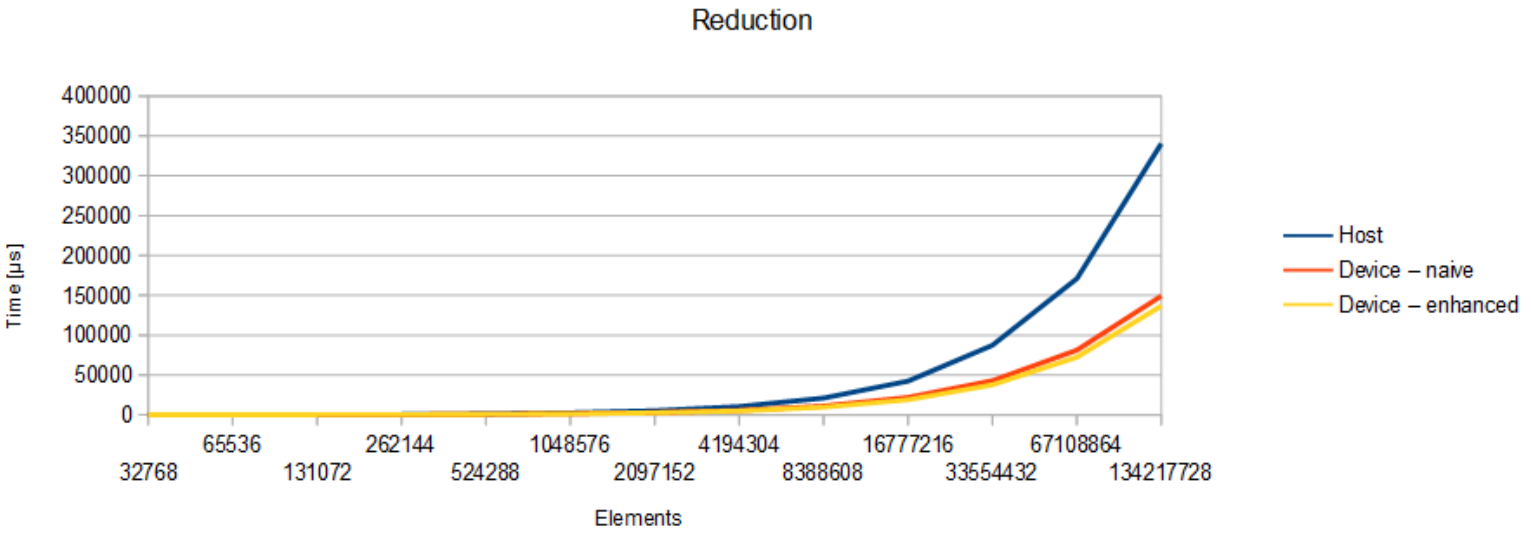
Our last lab was about reduction. Like in the last laboratory we were supposed to create a complete CUDA application. The purpose was to calculate the sum of 1D vector input data. If we want to do it in less time complexity then linear we have to parallel sum up two numbers reducing the length of the vector twice in each iteration. That gives log(n) time complexity. We made two programs, the first with a naive method and the with an enhanced method in which we better used available resources.

Results

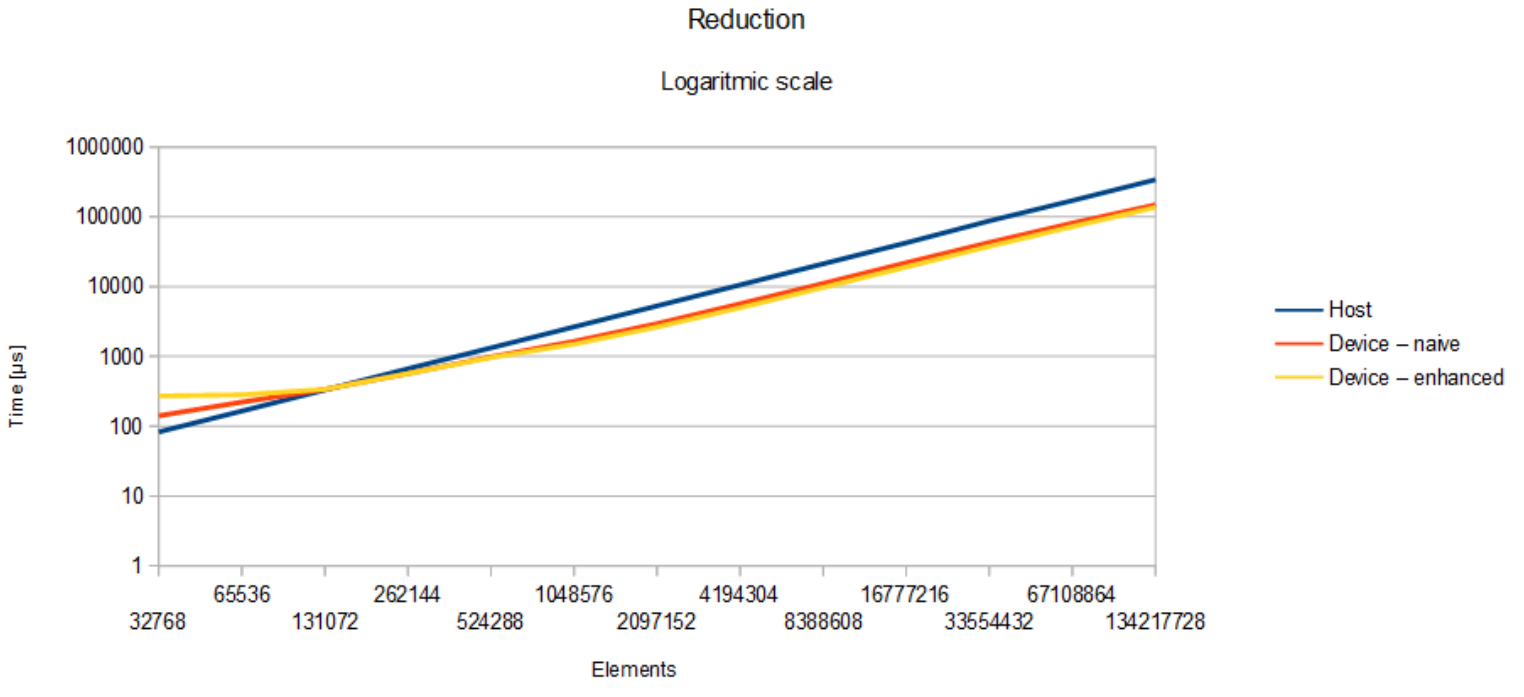
We used wide range of the input vector size. All measurements were made 10 times, then we obtained the avengare. All times are provided in microseconds (μs).

Elements	Host	Device - navie	Device - enhanced
32768	83	141,4	272
65536	165,1	222,1	282,3
131072	331,3	338,2	338,5
262144	666,3	568,6	568,2
524288	1326,5	986,8	967,6
1048576	2648,3	1649,3	1501,4
2097152	5296,5	2950,7	2642,5
4194304	10602,9	5677,9	4986,8
8388608	21231,7	11147,4	9738,4
16777216	42461,1	22005,1	19113,2
33554432	87424,2	42979,9	37758,7
67108864	171085,2	81376,7	72467,9
134217728	340552,9	149351,8	136989,4

Table 1. Comparison of results for both kernels and host-side processing.



Pic. 1. Graph from Table 1.



Pic. 2. Graph from Table 1. in logarithmic scale.

Conclusions

Once again CUDA improved the well known algorithm even using a very simple kernel. Hovever, naive method was ineffective. The main issues was keeping only part of the threads active, mismatch with the blocks grid and slow modulo operation. Here comes the enhanced method. We want to keep the active threads consecutive and process all of the iterations in the stride-grid technique using shared memory. At this point mapping is crucial as we also have to consider the divergence between the grid and the size of the data. As the last wrap of each block does not require synchronization, we unroll the last part of the loop. This technique can better use the shared memory, avoids modulo operation and provides more performance.