Introduction to CUDA and OpenCL
Lab 4 report

26.03.2020
Piotr Litwin
Paweł Skalny

Introduction

The fourth lab was about grid processing configuration methods, managed memory, error handling and first implementation of the matrix multiplication algorithm. We have gotten many code samples, which included mismatch and stride-grid techniques, managed memory, error wrapper, matrix multiplication for both strategies of memory handling. We had been obligated to analyse them and prepare own error wrapper.

Results and conclusions

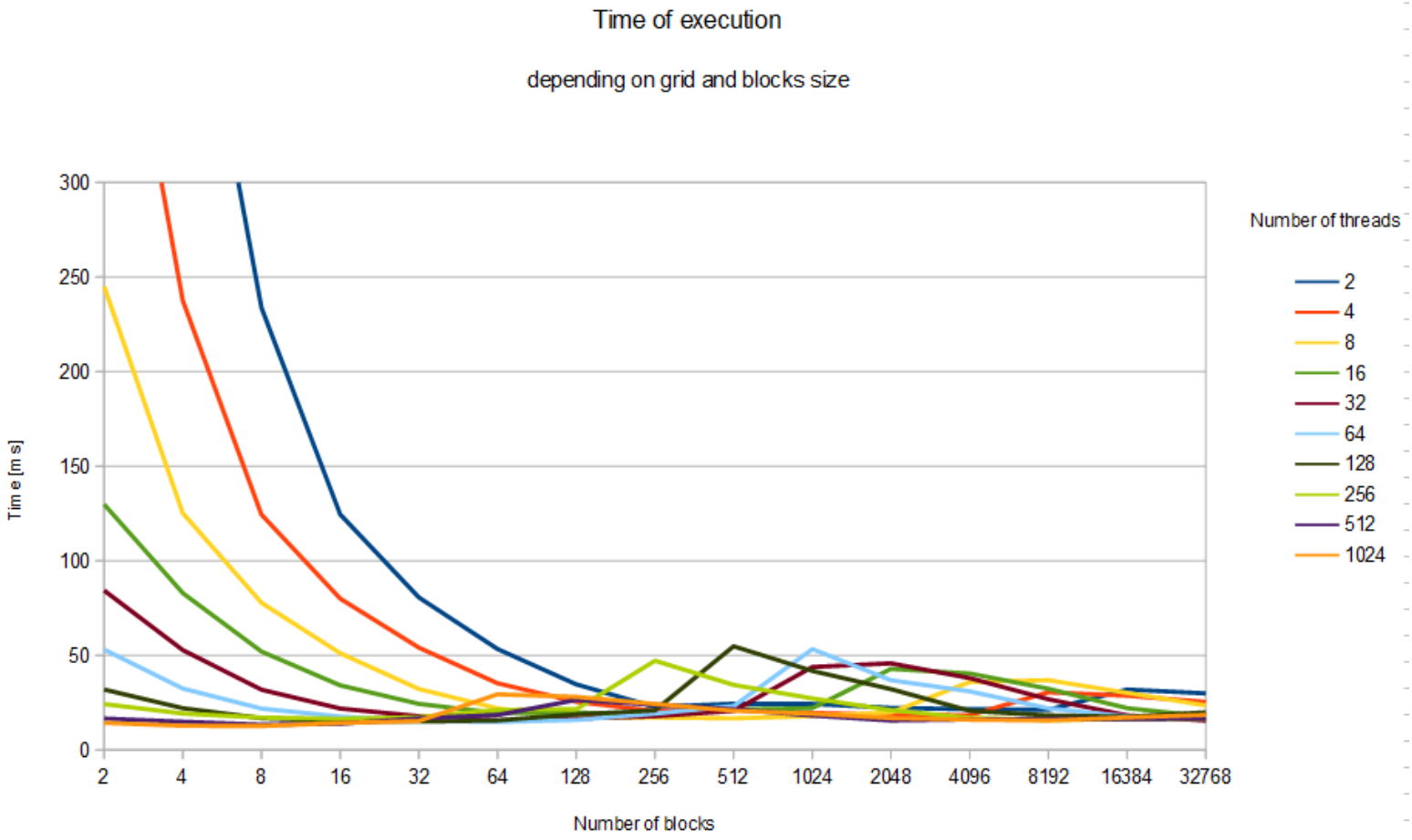
Firstly, we prepared the error wrapper - the simple function receiving the result of the Cuda API method and a message to print in the error case. It helps to organize code and increase transparency.

Analysing the mismatch and the stride-gride examples we concluded, that the first method is overall worse. In the mismatch, we have to define the dimensions of the grid depending on the data size. The use of the stride-grid technique allows us to keep the fixed size grid. If the logical threads are less than the data size requires, the mismatch technique will fail.

We created a program for processing configuration methods and then we wrote a complex script that used nvprof command to obtain execution time. We ran the program for various dimensions of blocks (rows) and grid(columns) ten times for each configuration (about 1500 measurements).

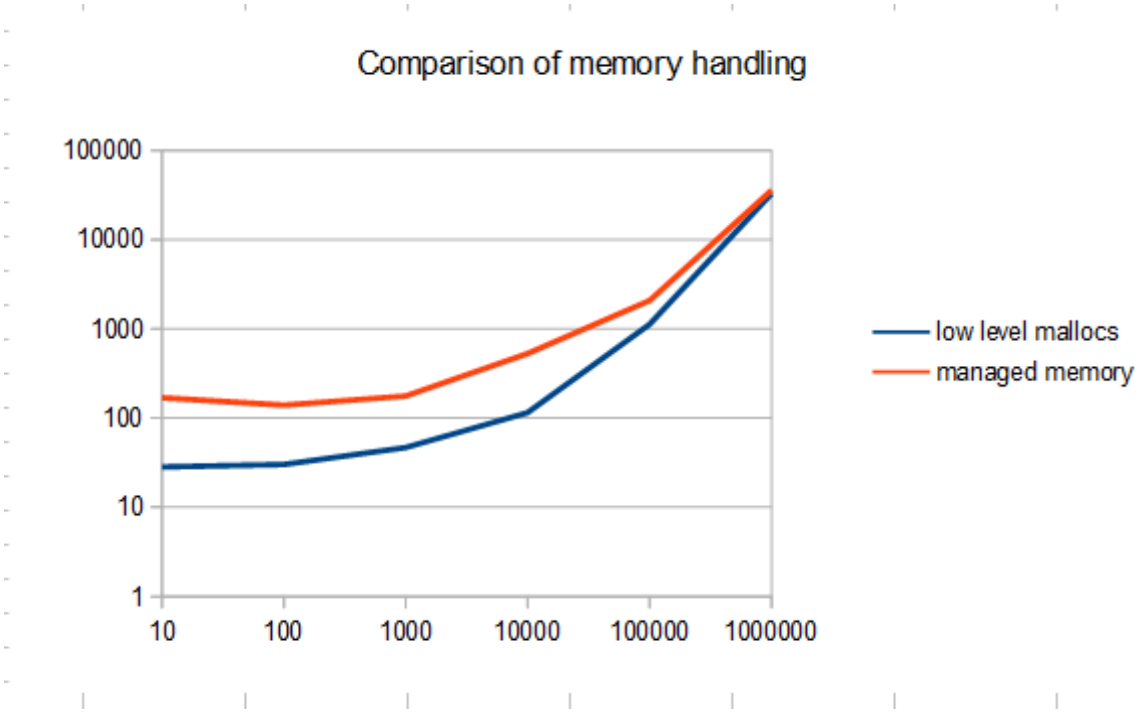
	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768
2	789,582	458,75	233,759	124,514	80,593	53,3791	34,7065	22,7144	24,4374	24,3406	22,273	21,5796	21,2514	31,9553	29,9336
4	467,315	237,526	124,357	79,9815	54,0713	35,2617	25,4968	20,5641	21,6692	19,5854	18,5613	18,3731	30,5092	28,663	25,3876
8	244,989	125,153	77,8021	51,1205	32,2309	22,0019	16,8921	17,1101	16,7149	17,853	19,8016	35,919	36,8678	30,1852	23,5606
16	129,816	82,9302	52,0045	34,1614	24,3802	19,4945	19,7232	18,9684	21,4156	22,1371	42,8454	40,4448	32,3592	22,1221	17,6951
32	84,3056	52,7889	31,8675	21,9048	17,8497	15,4818	16,6152	17,8176	20,6517	43,8639	45,7983	37,9839	26,7924	18,2765	15,5201
64	53,0494	32,4615	21,8909	17,646	15,383	14,8188	15,8455	19,2359	23,2102	53,3388	36,8645	30,9627	21,9878	17,7337	17,4455
128	31,9567	22,0858	16,7741	15,346	15,0453	15,6194	18,8225	20,9073	54,7931	41,7857	32,1699	20,7115	18,2408	17,5445	19,8719
256	24,2559	19,2553	17,1211	16,4402	17,3792	21,2049	21,5494	47,161	34,4032	27,2398	21,0119	16,9518	15,8145	16,2872	16,6231
512	16,6309	14,9045	13,3883	13,862	16,7765	18,4422	26,4648	23,9495	21,4622	18,1935	15,5526	16,1537	16,3464	16,3349	16,5705
1024	14,3332	12,8333	12,6401	14,2782	15,3372	29,4273	28,0852	24,3018	20,7072	19,151	16,835	16,0447	15,5549	17,1818	18,5136

Table 1. Time of execution of simple operation (multiplication by 2) for 10e7 elements, number of blocks – columns, threads for each block – rows.
*Time for mismatch technique – **19,86395** (200 measurements)*
All times provided in ms.



Pic. 1. Graph from Table 1.

Lastly, we focused on managed memory. Based on the provided samples we created a program, which calculate time of computing for different grid processing configuration, grid dimensions and memory management. The results are below:



Pic. 2. Time of execution for both low-level and managed memory in logarithmic scale

As we can see on *Pic. 2*, the conclusion is that low level malloc is a little faster, but it is hardly noticeable for larger data size. Furthermore managed memory is easer to use and we do not have to worry about synchronization with the host side memory.