Introduction to CUDA and OpenCL
Lab 6 report

16.04.2020
Piotr Litwin
Paweł Skalny

Introduction

During sixth lab we analyzed three different examples and we were trying to improve them. Firstly, we focused on multi-streaming. The technique that allows us to launch several kernels in parallel. We tested all possible configurations for the sample we were working on. Then we were trying do the best to improve Single-Precision A*X Plus Y operation. Finally, we improved provided heat transport simulation code using simple OpenACC macros.

Results and conclusions

We we given a sample of vector addition with streams implemented for vector initialization. We changed the program to execute with different configurations of streams added; no stream, 1 stream, 2 stream, stream on output, 3 stream, one for each vector. The table beneath shows times of execution of all these configurations.

	0 streams		a vec stream		a b vec streams		c vec stream		all streams	
Num of elements	Add vector	Init gpu	Add vector	Init gpu	Add vector	Init gpu	Add vector	Init gpu	Add vector	Init gpu
1024	204.98us	-	171.07us	3.7440us	202.78us	7.2320us	3.6160us	130.49us	3.6470us	10.752us
4096	406.83us	-	384.11us	3.7440us	194.56us	7.2960us	337.30us	3.7120us	3.7110us	10.752us
16384	789.44us	-	460.81us	3.8720us	272.47us	7.4230us	442.70us	3.8400us	4.0000us	10.976us
65536	1.8405ms	-	693.38us	4.2560us	363.03us	8.0630us	1.1136ms	4.1920us	4.8640us	11.776us
262144	3.2340ms	-	2.3301ms	5.2470us	1.2085ms	20.064us	2.3698ms	6.5280us	23.135us	20.031us
1048576	11.135ms	-	8.1938ms	26.399us	4.2824ms	57.629us	8.0690ms	25.375us	87.165us	84.988us
4194304	36.223ms	-	28.682ms	116.83us	13.445ms	255.00us	32.461ms	119.45us	341.65us	396.75us
16777216	122.29ms	-	94.723ms	517.64us	48.663ms	1.1136ms	134.82ms	569.70us	1.4099ms	1.8481ms

Summary:

From the table we see that adding streams quickens the execution of kernel significantly. The improvement of execution comes from the concurrency that the streams provide. The data from the vectors is provided while it is being initialized which is them being executed. The kernel doesn’t have to wait for the data to be fully initialized and can work on the batches of data provided from streams. Adding third stream doesn't make things better, because we don't have to initialise the c vector as long as it is the output vector.

SAXPY:

At the beginning we fixed a little bug in the provided code – the kernel received integers instead floating point numbers. We made couple of measurements and there are results:

Without any changes (just fix arguments): average **156 us**
Notice that we are looking for less than 25us, so there is much work to do.

Then we realised that we knew what exactly data will be needed in our saxpy function. So why dont use prefetching?

With prefetching data: average **122us**
Better, but not great.

Another idea was to use streams. We also rebuilt the saxpy kernel for the stride-grid technique.

With prefetching data, streams, and stride-grid: average **50us**
Really close.

Then we removed the unnecessary initialisation of the output vector: average **42us**
Still not enough.
Finally, we kept the stride-grid for the vector initialisation, but for the saxpy kernel we used a separate grid (mismatched). And voilà we got **2us –** very satisfactory.

Heat transport simulation:

This time we got the full algorithm code and one task – make it faster – quickly. We had to detect a loop with independent iterations, analyse the required data (memory management) and we were ready to add short OpenACC macros to magically speed up computing.

It shows how easily we can accelerate common programs written in pure C/C++ with minimal interference.