Tutorial № 3

JI Zhuoran, The University of Hong Kong 28/07/2018

We have seen vector addition and vector dot product. In this tutorial, we will learn the most important application of GPUs: the matrix multiplication. Matrix multiplication is the key operation of deep learning, screen render and VR. There are highly optimization library for matrix multiplication, such as cuBlas. However, we are going to use this example to learning:

- what is the meaning of "instructions per memory access"
- the use of dynamically-sized shared memory

Problem 1

In this problem, you will implemented a trivial matrix multiplication:

1. Read through **prac3a.cu** and finish the kernel code

Problem 2

The implementation in problem 1 does not perform well. The main reason the naive implementation doesn't perform so well is because we are accessing the GPU's off-chip memory way too much. Please count with me: to do the $M \times N \times K$ multiplications and additions, we need $M \times N \times 2$ loads and $M \times N$ stores. Since the multiplications and additions can actually be fused into a single hardware instruction (FMA), the computational intensity of the code is only 0.5 instructions per memory access (search for the Roofline model to find out why this is bad). Although the GPU's caches probably will help us out a bit, we can get much more performance by manually caching sub-blocks of the matrices (tiles) in the GPU's on-chip local memory (== shared memory in CUDA).
In this problem, we are going to sove the above problem.

1. Read the tutorial. Noted that the code in this tutorial in written in OpenCL, another GPUs programming language. (To be honest, OpenCL is more than GPUs programming language, see OpenCL if you are interested).
2. Read through the **prac3b.cu** and finished it by what you have learnt.
3. compile and compare the executaion time

