Numb_PyT_Lab

December 6, 2023

Massive parallel programming on GPUs and applications, by Lokman ABBAS TURKI

1 4. Add arrays with Numba jit and tensors multiplication with PyTorch

1.1 4.1 Objective

The main purpose of this lab is to show that there are other options for GPU parallelization that are conceptually very close to the use of CUDA API with C/C++ language. At the same time, this is an additional opportunity to become familiar with CUDA syntax by implementing simple examples.

With Python, multiple options to use GPUs are possible. The tensor library PyTorch is certainly the most used one. We can also use CuPy or Numba jit (Just-in-time compilation) for usual array computations. In this lab, we use both the Numba jit and PyTorch with documentation found at

Numba for CUDA GPUs

PYTORCH DOCUMENTATION

1.2 4.2 Content

First, we need to import

[122]:
```
from numba import cuda
import numpy as np
import time
import torch
```

1.2.1 4.2.1 Array addition with Numba jit

a) Define arrays a and b with a size of 100000000 values using function arange.

b) Calling function time.time(), compute the execution time of the addition on the host.

c) Using the documentation, write a kernel that adds an array a to an array b and puts the result in c.

d) Using function zeros, resets the values in array c.

e) Using to_device function transfer the values of a to aGPU, of b to bGPU, and of c to cGPU.

f) Execute the addition kernel with the appropriate number of threads and number of blocks, and compute its execution time.

1.2.2 4.2.2 Tensor multiplication with PyTorch

In the following questions, tensors are created and randomly filled using torch.rand.

a) Define tensors A and B on the host then perform the multiplication and compute the execution time.

b) Define tensors A and B on the device then perform the multiplication and compute the execution time.

[ ]: