ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

Федеральное государственное образовательное учреждение высшего профессионального образования ордена Трудового Красного Знамени

«МОСКОВСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ СВЯЗИ И ИНФОРМАТИКИ»

МКиИТ

**ЛАБОРАТОРНАЯ РАБОТА №6**

По дисциплине «Распределенные и параллельные вычисления»

На тему **«Стандартное произведение матриц»**

Выполнил студент 2 курса группы МБД2032:

Трояновский Семён Владиславович

Москва

2021

**Задача лабораторной работы:**

Реализовать простое произведение полностью заполненных матриц “в лоб”. В данной лабораторной производить оптимизации типа разбиения на блоки и использования разделяемой памяти не требуется.

**Ход выполнения:**

1. **Выбор инструмента**

Так как опыт работы с библиотекой numba в лабораторной работе #5, показался мне удачным, в этой лабораторной работе будет также использована эта библиотека.

1. **Код программы**

Для простого перемножения матриц достаточно выполнить следующий код:

from \_\_future\_\_ import division

from numba import cuda

import numpy

import math

# CUDA kernel

@cuda.jit

def matmul(A, B, C):

"""Perform matrix multiplication of C = A \* B

"""

row, col = cuda.grid(2)

if row < C.shape[0] and col < C.shape[1]:

tmp = 0.

for k in range(A.shape[1]):

tmp += A[row, k] \* B[k, col]

C[row, col] = tmp

# Host code

# Initialize the data arrays

A = numpy.full((24, 12), 3, numpy.float) # matrix containing all 3's

B = numpy.full((12, 22), 4, numpy.float) # matrix containing all 4's

# Copy the arrays to the device

A\_global\_mem = cuda.to\_device(A)

B\_global\_mem = cuda.to\_device(B)

# Allocate memory on the device for the result

C\_global\_mem = cuda.device\_array((24, 22))

# Configure the blocks

threadsperblock = (16, 16)

blockspergrid\_x = int(math.ceil(A.shape[0] / threadsperblock[0]))

blockspergrid\_y = int(math.ceil(B.shape[1] / threadsperblock[1]))

blockspergrid = (blockspergrid\_x, blockspergrid\_y)

# Start the kernel

matmul[blockspergrid, threadsperblock](A\_global\_mem, B\_global\_mem, C\_global\_mem)

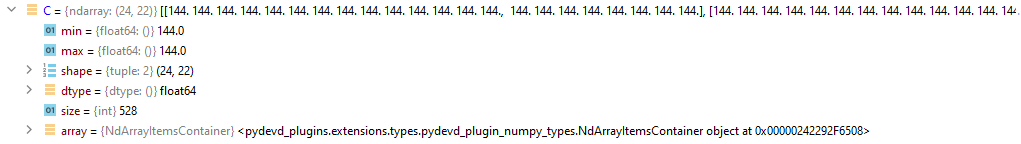
# Copy the result back to the host

C = C\_global\_mem.copy\_to\_host()

print(C)

1. **Результаты**

Результатом работы программы с заданными параметрами является такая матрица:



1. **Ответы на вопросы**

(1) Сколько операций с плавающей точкой будет произведено вашим ядром? Объясните.

*numCRows \* numCCols скалярных произведений = 2 \* numCRows \* numCCols \* numACols.*

(2) Сколько чтений из глобальной памяти будет произведено вашим ядром? Объясните.

*numCRows \* numCCols скалярных произведений = 2 \* numCRows \* numCCols \* numACols.*

(3) Сколько записей в глобальную память будет произведено вашим ядром? Объясните.

*По количеству выходных элементов матрицы. numCRows \* numCCols.*

(4) Опишите возможный вариант оптимизации, которую можно произвести для того, чтобы увеличить производительность вашего ядра.

*Разбить матрицу на блоки, помещая их в разделяемую память, чтобы снизить количество обращений к глобальной памяти.*

(5) Назовите три применения произведения матриц.

*Произведение матриц используется в почти каждом ресурсоемком приложении. Например, нейронные сети, компьютерная графика, дифференциальные уравнения в частных производных.*

1. **Вывод**

В данной работе я реализовал просто произведение матриц без задействования shared memory.