ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

Федеральное государственное образовательное учреждение высшего профессионального образования ордена Трудового Красного Знамени

«МОСКОВСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ СВЯЗИ И ИНФОРМАТИКИ»

МКиИТ

**ЛАБОРАТОРНАЯ РАБОТА №5**

По дисциплине «Распределенные и параллельные вычисления»

На тему **«Блочное перемножение матриц на CUDA»**

Выполнил студент 2 курса группы МБД2032:

Трояновский Семён Владиславович

Москва

2021

**Задача лабораторной работы:**

Реализовать блочное произведение матриц с использованием разделяемой памяти.

**Ход выполнения:**

1. **Выбор библиотеки**

В предыдущих лабораторных работах была использована библиотека PyCuda, однако

Для выполнения данной задачи была выбрана для разнообразия Python библиотека numba. Она также предоставляет удобное высокоуровневое API. Используя эту библиотеку можно взаимодействовать с ядром без написания кода на C++.

1. **Полный код**

Полный код для умножения матриц с использованием разделяемой памяти выглядит так:

import numpy as np

from numba import cuda, float32

@cuda.jit

def fast\_matmul(A, B, C):

# Define an array in the shared memory

# The size and type of the arrays must be known at compile time

TPB = N

sA = cuda.shared.array(shape=(TPB, TPB), dtype=float32)

sB = cuda.shared.array(shape=(TPB, TPB), dtype=float32)

x, y = cuda.grid(2)

tx = cuda.threadIdx.x

ty = cuda.threadIdx.y

bpg = cuda.gridDim.x # blocks per grid

if x >= C.shape[0] and y >= C.shape[1]:

# Quit if (x, y) is outside of valid C boundary

return

# Each thread computes one element in the result matrix.

# The dot product is chunked into dot products of TPB-long vectors.

tmp = 0.

for i in range(bpg):

# Preload data into shared memory

sA[tx, ty] = A[x, ty + i \* TPB]

sB[tx, ty] = B[tx + i \* TPB, y]

# Wait until all threads finish preloading

cuda.syncthreads()

# Computes partial product on the shared memory

for j in range(TPB):

tmp += sA[tx, j] \* sB[j, ty]

# Wait until all threads finish computing

cuda.syncthreads()

C[x, y] = tmp

# This part is for initializing everything

M = 128

N = 32

a = np.ones(M\*N).reshape(M,N).astype(np.float32)

b = np.ones(M\*N).reshape(N,M).astype(np.float32)

c = np.zeros((M, M)).astype(np.float32)

d\_a = cuda.to\_device(a)

d\_b = cuda.to\_device(b)

d\_c = cuda.to\_device(c)

block\_size = (N,N)

grid\_size = (int(M/N),int(M/N))

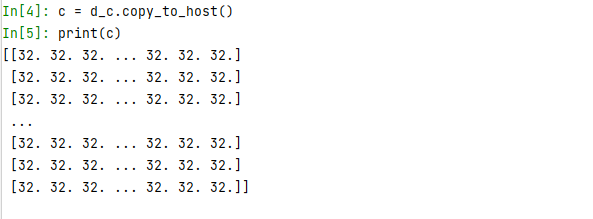
fast\_matmul[grid\_size,block\_size](d\_a, d\_b, d\_c)

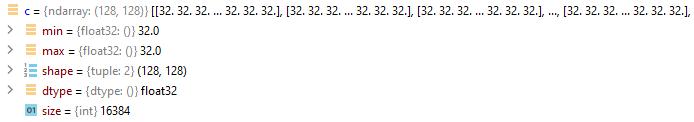
c = d\_c.copy\_to\_host()

print(c)

1. **Результат**

Результат выполнения программы выглядит так:





Что является верным перемножением матриц.

1. **Ответы на вопросы**

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

*Одно скалярное произведение на один элемент выходной матрицы: 2 \* numACols \* numCRows \* numCCols*

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

*Каждый поток произведет 2\*ceil(numACol s⁄T ILE\_W IDTH) чтений, а задействованных потоков всего numCCols \* numCRows.*

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

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

(4) Опишите, какие дальнейшие оптимизации помогли бы вам улучшить производительность ядра.

*Настройка параметров запуска ядра для оптимизации размещения в памяти.*

(5) Сравните сложности реализации этой и предыдущей лабораторной. С какими сложностями вы столкнулись в данной лабораторной?

*Ошибки можно сделать при переводе из глобального адресного пространства в адресное пространство разделяемой памяти.*

(6) Представим: имеются матрицы, размерность которых превышает максимальную размерность нитей. Набросайте алгоритм,

который выполняет перемножение этих матриц.

*Каждый поток должен обрабатывать 4, 9 или 16 соседних элементов матрицы.*

(7) Представим: имеются матрицы, которые не умещаются в глобальной памяти. Набросайте алгоритм, который выполняет произведение таких матриц.

*По аналогии с блочным перемножением с использованием разделяемой памяти нужно произвести разбиение на блоки, но на стороне хоста.*

1. **Вывод**

В данной работе я реализовал блочное произведение матриц с использованием разделяемой памяти, а также попробовал на практике альтернативный вариант бибилотеки на Python для работы с GPU CUDA.