ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

Федеральное государственное образовательное учреждение высшего профессионального образования ордена Трудового Красного Знамени

«МОСКОВСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ СВЯЗИ И ИНФОРМАТИКИ»

МКиИТ

**ЛАБОРАТОРНАЯ РАБОТА №4**

По дисциплине «Распределенные и параллельные вычисления»

На тему **«Сложение векторов на CUDA»**

Выполнил студент 2 курса группы МБД2032:

Трояновский Семён Владиславович

Москва

2021

**Задача лабораторной работы:**

Цель данной лабораторной – познакомить слушателей с CUDA API, реализовав сложение векторов.

**Ход выполнения:**

1. **Алгоритм расчетов**

Для начала напишем алгоритм расчётов, который является основной частью программы.

В данном случае он выглядит так:



1. **Полный код программы**

В данном случае нам также нужно будет перемещать данные между CPU и GPU, инициализировать объекты и выполнять другие операции. Также, в конце программы добавлен тест для проверки результата. Скрипт выглядит так:

import numpy as np

# --- PyCUDA initialization

import pycuda.driver as cuda

import pycuda.autoinit

from pycuda.compiler import SourceModule

###################

# iDivUp FUNCTION #

###################

def iDivUp(a, b):

return a // b + 1

########

# MAIN #

########

start = cuda.Event()

end = cuda.Event()

N = 100000

BLOCKSIZE = 256

# --- Create random vectorson the CPU

h\_a = np.random.randn(1, N)

h\_b = np.random.randn(1, N)

# --- Set CPU arrays as single precision

h\_a = h\_a.astype(np.float32)

h\_b = h\_b.astype(np.float32)

# --- Allocate GPU device memory

d\_a = cuda.mem\_alloc(h\_a.nbytes)

d\_b = cuda.mem\_alloc(h\_b.nbytes)

d\_c = cuda.mem\_alloc(h\_a.nbytes)

# --- Memcopy from host to device

cuda.memcpy\_htod(d\_a, h\_a)

cuda.memcpy\_htod(d\_b, h\_b)

mod = SourceModule("""

#include <stdio.h>

\_\_global\_\_ void deviceAdd(float \* \_\_restrict\_\_ d\_c, const float \* \_\_restrict\_\_ d\_a, const float \* \_\_restrict\_\_ d\_b, const int N)

{

const int tid = threadIdx.x + blockIdx.x \* blockDim.x;

if (tid >= N) return;

d\_c[tid] = d\_a[tid] + d\_b[tid];

}

""")

# --- Define a reference to the \_\_global\_\_ function and call it

deviceAdd = mod.get\_function("deviceAdd")

blockDim = (BLOCKSIZE, 1, 1)

gridDim = (iDivUp(N, BLOCKSIZE), 1, 1)

start.record()

deviceAdd(d\_c, d\_a, d\_b, np.int32(N), block = blockDim, grid = gridDim)

end.record()

end.synchronize()

secs = start.time\_till(end) \* 1e-3

print("Processing time = %fs" % (secs))

# --- Copy results from device to host

h\_c = np.empty\_like(h\_a)

cuda.memcpy\_dtoh(h\_c, d\_c)

if np.array\_equal(h\_c, h\_a + h\_b):

print("Test passed!")

else :

print("Error!")

# --- Flush context printf buffer

cuda.Context.synchronize()

1. **Результат**

Таким образом, по выполнении скрипта в консоли мы ожидаем увидеть подобный вывод:

****

Это означает, что программа отработала верно.

1. **Ответы на вопросы**

(1) Сколько операций с плавающей точкой выполняется в вашем ядре сложения векторов? ОБЪЯСНИТЬ.

*N – по одному на каждую пару элементов входных векторов.*

(2) Сколько чтений из глобальной памяти производится вашим ядром? ОБЪЯСНИТЬ.

*2N – по одному на каждый элемент каждого входного вектора.*

(3) Сколько записей в глобальную память производится вашим ядром? ОБЪЯСНИТЬ.

*N - по одному на каждый элемент результирующего вектора.*

(4) Опишите возможный вариант оптимизации ядра для увеличения производительности.

*Разбить на несколько ядер, чтобы перекрыть передачу данных и исполнение ядра.*

(5) Назовите три приложения сложения векторов.

*Анализ больших данных, статистика, быстрое преобразование Фурье.*

1. **Вывод**

В данной работе я реализовал сложение векторов с использованием GPU и в ходе работы лучше познакомился с CUDA API.