МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Информационные технологии и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №1
по курсу «Программирование графических процессоров»

Освоение программного обеспечения для работы с технологией CUDA. Примитивные операции над векторами.

Выполнил: И.Д. Черненко
Группа: 8О-406Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2021

Условие

1. **Цель работы:** Ознакомление и установка программного обеспечения для работы с программно-аппаратной архитектурой параллельных вычислений(CUDA). Реализация одной из примитивных операций над векторами.
2. **Вариант задания: Вариант 1**
   Входные данные: На первой строке задано число n -- размер векторов. В следующих 2-х строках, записано по n вещественных чисел -- элементы векторов.
   Выходные данные: Необходимо вывести n чисел -- результат сложения исходных векторов.

Программное и аппаратное обеспечение

Compute capability : 6.1
Name : GeForce GTX 1050
Total Global Memory : 2096103424
Shared memory per block : 49152
Registers per block : 65536
Max threads per block : (1024, 1024, 64)
Max block : (2147483647, 65535, 65535)
Total constant memory : 65536
Multiprocessors count : 5

OS:Linux
compiler:nvcc
IDE:Clion
Editor:nano

Метод решения

Для решения данной задачи не потребовалось какого-то особого алгоритма, просто было выделено три массива с типом double в куче. А также с помощью **cudaMalloc** было выделено место под три массива на устройстве. Далее с помощью **cudaMemcpy** данные были переданы с хоста на устройство. После этого на устройстве были выполнены необходимые операции, а именно с помощью цикла **for** было произведено сложение двух векторов и данные были переданы обратно на хост и выдан результат, как и освобождена память на устройстве и хосте.

Описание программы

Вся программа была реализована в одном файле **main.cu**.

Реализованное ядро:

```
__global__ void sum_vectors(double* vector1, double* vector2, double* sum,
int n) {
    int i, idx = blockDim.x * blockIdx.x + threadIdx.x;
    int offset = blockDim.x * gridDim.x;
    for(i = idx; i < n; i += offset)
        sum[i] = vector1[i] + vector2[i];
}
```

Как и было указано в задании в качестве вещественного типа данных был использован тип **double**, а в качестве типа данных для определения размера массивов был использован тип **int**, который подошел под ограничение размера массивов $\mathbf{n < 2^{25}}$.

Результаты

1. Замеры времени работы с **CUDA**

Размер одного вектора \ Конфигурации ядер(размерность сетки блоков, размер блока)	1,32	256,256	512,512	1024,1024
10000	201.22us	25.661us	87.322us	424.40us
50000	997.16us	40.942us	103.11us	445.29us
100000	1.9853ms	71.922us	122.02us	477.47us
200000	3.9933ms	141.00us	160.07us	530.70us
500000	10.056ms	349.39us	347.94us	695.38us
1000000	19.793ms	698.08us	694.07us	971.76us

2. Замеры времени работы без **CUDA**

Размер одного вектора	Время
10000	0.085447 ms
50000	0.302256 ms
100000	0.489135 ms
200000	0.961888 ms
500000	2.16174 ms
1000000	4.29754 ms

Как можно заметить с использованием **CUDA**, производительность в большинстве случаев растет в несколько раз по сравнению с вычислениями на **CPU,** при этом при увеличении размера входных данных разница становится намного заметнее. Кроме того, можно заметить что при увеличении размерности сетки блоков и самих блоков

производительность начинает падать, что заметно особенно на малых тестах. Вероятнее всего это происходит из-за задержки при их синхронизации при больших размерах блоков(https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html). Интересным наблюдением является и то, что при малой размерности сетки производительность даже хуже чем на **CPU**, причиной является то, что при такой конфигурации задействованы не все мультипроцессоры.

Выводы

Алгоритм используемый в данной задаче, является весьма базовым, но при этом он может помочь при решении задач линейной алгебры и компьютерной графики. Сам процесс программирования был совершенно не сложным, проблемы возникли только из-за собственной невнимательности, но благодаря удобным мануалам по **CUDA**, реализация не вызвала особых проблем. Полученные результаты прежде всего говорят о полезности использования **CUDA ядер** при различных вычислениях, так как благодаря использованию блоков и соответственно нитей, из которых они состоит, мы можем распараллелить наши вычисления по блокам и потокам, что и ведёт к значительному увеличению производительности.