МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки 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: *А.А. Почечура*
Группа: *8О-406Б*
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2023

Условие

Цель работы. Ознакомление и установка программного обеспечения для работы с программно-аппаратной архитектурой параллельных вычислений(CUDA). Реализация одной из примитивных операций над векторами. В качестве вещественного типа данных необходимо использовать тип данных *double*. Все результаты выводить с относительной точностью 10^{-10}. Ограничение: $n < 2^{25}$.

Вариант 2. Вычитание векторов.

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 вычитания исходных векторов.

Программное и аппаратное обеспечение

Графический процессор:

Compute capability: 7.5
Name: Tesla T4
Total Global Memory: 15835398144
Shared memory per block: 49152
Registers per block: 65536
Warp size: 32
Max threads per block: (1024, 1024, 64)
Max block: (2147483647, 65535, 65535)
Total constant memory: 65536
Multiprocessors count: 40

Процессор:

vendor_id : GenuineIntel
cpu family: 6
model: 85
model name: Intel(R) Xeon(R) CPU @ 2.00GHz
stepping: 3
microcode: 0xffffffff
cpu MHz: 2000.184
cache size : 39424 KB
physical id: 0
siblings: 2
core id: 0

cpu cores : 1
apicid: 0
initial apicid: 0
fpu: yes
fpu_exception: yes
cpuid level: 13
wp: yes
bogomips : 4000.36
clflush size: 64
cache_alignment : 64
address sizes: 46 bits physical, 48 bits virtual

Оперативная память и жёсткий диск:

Memory:
description: System memory
physical id: 0
size: 13GiB

Mem**:** 12982.6 MiB
Storage**:** 55 GiB

Программное обеспечение:

Google Colab
Операционная система: Ubuntu 22.04.2 LTS
Оболочка: python3
inxi: 3.3.13

Метод решения

Чтобы получить результат поэлементного вычитания двух векторов, используя распараллеливание действий по потокам на графическом процессоре, для начала нам потребуется считать эти вектора со стандартного ввода, а затем передать их на графический процессор. Затем, на графическом процессоре с помощью функции ядра мы вычтем поэлементно из первого вектора второй вектор соответственно (каждому потоку за раз мы подаём ровно одну операцию вычитания). Таким образом, первый вектор будет являться ответом на задачу. Затем передаём первый вектор с графического процессора обратно на ЦПУ и выводим его в качестве результата.

Описание программы

Считаем вектора со стандартного ввода и назовём их *arr1* и *arr2*. Перенесём их на графический процессор с помощью функции *cudaMemcpy* (для этого заведём отдельные вектора *dev_arr1* и *dev_arr2*). Затем запустим функцию ядра *kernel*, которой передадим в качестве аргументов вектора *dev_arr1*, *dev_arr2* и их размер *n*. Для работы ядру выделим 512 блоков, в каждом из которых будет 512 потоков. В переменной *idx* вычислим номер текущего потока, в переменной *offset* посчитаем общее количество потоков. Далее на текущему потоке будем вычислять разность соответствующих элементов двух векторов, индекс которых равен *idx*, *idx + offset*, *idx + 2 * offset* и т.д., пока индекс меньше размера векторов *n*. Пробегаться по индексам со смещение будем для обеспечения более оптимально работы потоков (если бы мы разделили вектора на равные подряд идущие промежутки и двигались бы по каждому из них в отдельном потоке, это было бы гораздо медленнее). Результаты вычислений будем записывать в вектор *arr1*. Далее, с помощью функции *cudaMemcpy* переносим вектор *div_arr1* с графического процессора на ЦПУ и записываем его в *arr1*. Потом выводим массив *arr1* в качестве результата и очищаем память, выделенную под вектора на ЦПУ и графическом процессоре, с помощью функций *free()* и *cudafree()*.

Результаты

Количество блоков и потоков в блоке	Размер входных данных	n = 15	n = 1000	n = 1000.00	n = 1000.00.00
<< 1, 32 >>		2.5760000572e-02 мс	4.6560000628e-02 мс	1.4777920246e+00 мс	1.8812976074e+02 мс
<< 4, 64 >>		4.3391998857e-02 мс	4.8960000277e-02 мс	2.0326399803e-01 мс	2.6181407928e+01 мс
<< 32, 128 >>		2.4415999651e-02 мс	2.8127999976e-02 мс	3.1295999885e-02 мс	2.0222399235e+00 мс
<< 128, 128 >>		2.3328000680e-02 мс	2.6623999700e-02 мс	3.8304001093e-02 мс	1.0120960474e+00 мс
<< 256, 256 >>		2.6208000258e-02 мс	2.4415999651e-02 мс	2.5119999424e-02 мс	9.7286397219e-01 мс

<< 512, 512 >>	2.8640000150e-02 мс	3.6896001548e-02 мс	4.5216001570e-02 мс	9.6422398090e-01 мс
<< 1024, 1024 >>	3.7344001234e-02 мс	4.5088000596e-02 мс	5.8463998139e-02 мс	9.5276802778e-01 мс

Можно заметить, что при увеличении входных данных и малом количестве потоков время работы программы увеличивается довольно быстро. При увеличении количества потоков в один момент разница в работе становится практически незначительной (например, случаи << 512, 512 >> и << 1024, 1024 >>). Увеличение количества потоков также уменьшает рост времени работы в зависимости от увеличения входных данных.

Размер входных данных	Вид процессора	ГПУ (<< 512, 512 >>)	ЦПУ
n = 15		2.8640000150e-02 мс	2.0000000000e-03 мс
n = 1000		3.6896001548e-02 мс	4.0000000000e-03 мс
n = 1000.00		4.5216001570e-02 мс	3.4100000000e-01 мс
n = 1000.00.00		9.6422398090e-01 мс	9.1581000000e+01 мс

Нетрудно сделать вывод, что время работы программы на ГПУ на несколько порядков меньше времени работы на ЦПУ. Связано это с распределением процессов на потоки, что позволяет выполнять действия параллельно и, тем самым, экономить время.

Выводы

В данной лабораторной работе я впервые взаимодействовал с графическим процессором на программном уровне. Я познакомился с тем, как передавать данные с ЦПУ на ГПУ и получать их с ГПУ на ЦПУ в коде, как обращаться по индексу к потоку, научился писать функцию ядра, понял, как производить замер времени работы программы на ГПУ. Также в ходе работы я узнал, что вычисление результатов на ГПУ

сильно быстрее, чем на ЦПУ, за счёт распараллеливания вычислений. Полученные мной знания помогут мне в дальнейшем при выполнении следующих лабораторных работ.