

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»

Изучение технологии CUDA

Выполнил: *И. П. Попов*

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 1. Сложение 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 сложения исходных векторов.

Программное и аппаратное обеспечение

Дать характеристики графического процессора (compute capability, графическая память, разделяемая память, константная память, количество регистров на блок, максимальное количество блоков и нитей, количество мультипроцессоров), процессора, оперативной памяти и жесткого диска. Описать программное обеспечение (OS, IDE, compiler и тд.).

GP Tesla T4

Compute capability: 7.5

Dedicated video memory: 15835 MB

Shared memory per block: 49152 bytes

Register per block: 65536 bytes

Total constant memory: 65536 bytes

Max threads per multiprocessor: 2048

Max threads per block: 1024

Multiprocessors const: 40

CPU AMD Ryzen 3600X

Physical cores: 6

Threads: 12

Base clock speed: 3.8 GHz

Boost clock speed: 4.4 GHz

L1 cache: 384KB (per core)

L2 cache: 512KB (per core)

L3 cache: 32 MB (shared)

Chip lithography: 7 nm

16 Гб RAM

1 Тб HDD

OS – Windows 11 ProWSL, IDE – VS Code, compiler - nvcc

Метод решения

Считываем входные векторы(CPU), копируем их в CUDA, производим их сложение с использованием параллельных вычислений, результат передаем обратно на CPU.

Описание программы

Разделение по файлам, описание основных типов данных и функций. Обязательно описать реализованные ядра.

1. На вход программе поступает число n , которое определяет размер каждого вектора.
2. Мы резервируем память на CPU для трех векторов размера n и заполняем два из них с помощью потокового ввода.
3. Затем выделяем память для этих векторов на графическом процессоре (GPU) с использованием функции `cudaMalloc()`.
4. Данные из исходных векторов копируются на GPU с помощью функции `cudaMemcpy()`.
5. Запускается ядро - функция `kernel()`, которая имеет спецификатор вызова функции `global`, что означает, что функция вызывается с CPU, но выполняется на GPU.
6. Внутри функции `kernel()` вычисляются идентификатор текущей нити (`index`) и смещение (`offset`). Затем в цикле выполняется вычисление суммы элементов векторов.
7. Результат вычислений копируется обратно с GPU на CPU с использованием функции `cudaMemcpy()`.
8. Полученный вектор выводится на экран.
9. Происходит освобождение памяти.

```
__global__ void kernel(double* v1, double* v2, double* ans, int n) {  
    int offset = gridDim.x * blockDim.x;  
    int idx = blockDim.x * blockIdx.x + threadIdx.x;  
    for(int i = idx; i < n; i = i + offset){  
        ans[i] = v1[i] + v2[i];  
    }  
    return;  
}
```

Результаты

1. Отобразить в виде таблички или графиков замеры времени работы ядер с различными конфигурациями (начиная с `<<< 1, 32 >>>` и как минимум до `<<< 1024, 1024 >>>`) и различными входными данными (небольшие тесты, средние и предельные). **Обязательно указать единицы измерения времени.**

2. Произвести сравнение с CPU (для этого нужно реализовать свой вариант ЛР без использования технологии CUDA). Время на копирование входных-выходных данных не учитывать, замерять только время работы самого алгоритма.
3. Если программа подразумевает работу с изображениями, то необходимо наличие скриншотов.

Конфигурация/ входные данные	10^2	10^4	10^6	10^7
<1,32>	0.034464	0.635392	62.582527	624.777039
<32, 32>	0.031200	0.057088	4.209984	42.059040
<256, 256>	0.038144	0.033728	1.658688	18.557600
<512, 512>	0.050752	0.048672	1.611936	17.269823
<1024, 1024>	0.164256	0.164928	1.742656	16.433216
cpu	0.0	2.043580	5.027455	112.047385

Выводы

Обычно, GPU используется для работы с большими объемами данных, в то время как CPU лучше справляется с более сложными операциями над небольшими объемами данных. Выбор между CPU и GPU зависит от конкретных требований задачи и производительности системы.