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»

Изучение технологии CUDA

Выполнил: В.А. Петросян
Группа: 8О-408Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2020

Условие

Цель работы: Ознакомление и установка программного обеспечения для работы с программно-аппаратной архитектурой параллельных вычислений(CUDA). Реализация одной из примитивных операций над векторами.
Вариант № 2 "Вычитание векторов"

Программное и аппаратное обеспечение

GPU:

1. Compute capability: 3.0
2. Графическая память: 2147155968
3. Разделяемая память: 49152
4. Константная память: 65536
5. Количество регистров на блок: 65536
6. Максимальное количество блоков: (65535, 65535, 65535)
7. Максимальное количество нитей: (1024, 1024, 64)
8. Количество мультипроцессоров: 6

Сведения о системе:

1. Процессор: Intel Core i7-Q720 1.60GHz
2. Память: 8 ГБ
3. HDD: 465 ГБ

Программное обеспечение:

1. OS: Windows 7
2. IDE: Visual Studio 2019
3. Компилятор: nvcc

Метод решения

Когда количество потоков больше чем n каждый поток выполняет расчет не более одной позиции в результирующем массиве. Если же количество потоков меньше чем n, то некоторым потокам приходится вычислять более одного значения. Чтобы всё работало эффективно я производил смещение как было показано на лекции
(1, 2, 3, … , n, 1, 2, 3, …)

Описание программы

```
#include "cuda_runtime.h"
#include "device_launch_parameters.h"
#include <cstdlib>
#include <iostream>
#include <iomanip>

using namespace std;
typedef long long ll;
```

```
__global__ void kernel(double* a, double* b, double* answer, ll n)
{
    ll offset = gridDim.x * blockDim.x;
    ll idx = blockDim.x * blockIdx.x + threadIdx.x;
    for(ll i = idx; i < n; i = i + offset){
        answer[i] = a[i] - b[i];
    }
    return;
}

int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    ll n;
    cin >> n;

    double* a = new double[n];
    double* b = new double[n];
    double* answer = new double[n];

    for (ll i = 0; i < n; ++i) {
        cin >> a[i];
    }
    for (ll i = 0; i < n; ++i) {
        cin >> b[i];
    }
    double* nums1;
    double* nums2;
    double* res;
    if (cudaMalloc((void**)&nums1, sizeof(double)*n) != cudaSuccess) {
        cerr << "ERROR!   46\n";
        abort();
    }
    if (cudaMalloc((void**)&nums2, sizeof(double)*n) != cudaSuccess) {
        cerr << "ERROR!   50\n";
        abort();
    }
    if (cudaMalloc((void**)&res, sizeof(double) * n) != cudaSuccess) {
        cerr << "ERROR!   54\n";
        abort();
    }
```

```
    if (cudaMemcpy(nums1, a, sizeof(double)*n, cudaMemcpyHostToDevice)
                    != cudaSuccess) {
        cerr << "ERROR!   59\n";
        abort();
    }
    if (cudaMemcpy(nums2, b, sizeof(double) * n, cudaMemcpyHostToDevice)
                    != cudaSuccess) {
        cerr << "ERROR!   63\n";
        abort();
    }
    kernel <<<256, 256>>> (nums1, nums2, res, n);
    if (cudaGetLastError() != cudaSuccess) {
        cerr << "ERROR!   70\n";
        abort();
    }
    if(cudaMemcpy(answer, res, sizeof(double)*n, cudaMemcpyDeviceToHost)
                  != cudaSuccess) {
        cerr << "ERROR!   75\n";
        abort();
    }
    cout.precision(10);
    cout.setf(ios::scientific);
    for (ll i = 0; i < n; ++i) {
        cout << answer[i] << " ";
    }
    cout << "\n";
    if (cudaFree(nums1) != cudaSuccess) {
        cerr << "ERROR!   87\n";
        abort();
    }
    if (cudaFree(nums2) != cudaSuccess) {
        cerr << "ERROR!   91\n";
        abort();
    }
    if (cudaFree(res) != cudaSuccess) {
        cerr << "ERROR!   95\n";
        abort();
    }
    delete[] a;
    delete[] b;
    delete[] answer;
    return 0;
}
```

Результаты

CPU

N	10^5	10^6	10^7	$3 * 10^7$	$6 * 10^7$	$9 * 10^7$	10^8
Time	0.016	0.094	0.811	2.808	5.561	7.163	9.766

Конфигурация ядра <<1, 32>>

N	10^5	10^6	10^7	$3 * 10^7$	$6 * 10^7$	$9 * 10^7$	10^8
Time	0.043	0.066	0.285	0.788	1.589	2.312	2.565

Конфигурация ядра <<32, 32>>

N	10^5	10^6	10^7	$3 * 10^7$	$6 * 10^7$	$9 * 10^7$	10^8
Time	0.041	0.046	0.104	0.232	0.439	0.649	0.721

Конфигурация ядра <<64, 64>>

N	10^5	10^6	10^7	$3 * 10^7$	$6 * 10^7$	$9 * 10^7$	10^8
Time	0.041	0.042	0.097	0.215	0.402	0.589	0.658

Конфигурация ядра <<128, 128>>

N	10^5	10^6	10^7	$3 * 10^7$	$6 * 10^7$	$9 * 10^7$	10^8
Time	0.041	0.042	0.098	0.207	0.401	0.588	0.651

Конфигурация ядра <<256, 256>>

N	10^5	10^6	10^7	$3 * 10^7$	$6 * 10^7$	$9 * 10^7$	10^8
Time	0.041	0.042	0.098	0.207	0.401	0.588	0.651

Конфигурация ядра <<1024, 1024>>

N	10^5	10^6	10^7	$3 * 10^7$	$6 * 10^7$	$9 * 10^7$	10^8
Time	0.041	0.042	0.098	0.207	0.401	0.588	0.651

Выводы

Этот алгоритм применяется повсеместно как часть более сложных алгоритмов. На лекции большая часть трудных мест была разобрана, так что сложность программирования низкая. Так как использую видеокарту удалённо через ssh, то в следующих лабораторных работах отладка будет не простой. Идея записать номер строки в cerr не плоха, но даже она мне не поможет. По результатам из таблиц видно, что решая задачу, которая хорошо распараллеливается можно ускорить время выполнения программы в 15 раз.