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. Примитивные операции над векторами.

Выполнил: Симонов С.Я.
Группа: 8О-406Б-18
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2021

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:

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

Сведения о системе:

Операционная система: Ubuntu 14.04 LTS 64-x битная
Рабочая среда: nano
Компилятор: nvcc

Метод решения

Сперва выделяем память на устройстве, чтобы считать изначальные вектора, затем выделяем память на видеокарте. Копируем наши вектора с устройства на gpu. Запускаем функцию kernel, состоящей из заданного нами количества блоков и потоков. Функция обрабатывает наши значения в многопоточном режиме, причем каждый поток обрабатывает не последовательность элементов вектора идущих подряд, а расположенные на равном расстоянии друг от друга (смещение равно количеству потоков). Копируем Полученный вектор с gpu на cpu. Выводим полученный результат, чистим память.

Описание программы

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
__global__ void dif(double* a, double* b, double* c, int N) {
	int i, idx = blockDim.x * blockIdx.x + threadIdx.x;
	int offset = blockDim.x * gridDim.x;
	for (i = idx; i < N; i += offset) {
		c[i] = a[i] - b[i];
	}
}
```

```
int main(void) {
	clock_t start = clock();
	int N;
	scanf("%d", &N);
	double *a = (double*) malloc(sizeof(double) * N);
	double *b = (double*) malloc(sizeof(double) * N);
	double *c = (double*) malloc(sizeof(double) * N);
	double *dev_a, *dev_b, *dev_c;
	cudaMalloc((void**)&dev_a, sizeof(double) * N);
	cudaMalloc((void**)&dev_b, sizeof(double) * N);
	cudaMalloc((void**)&dev_c, sizeof(double) * N);
	for(int i = 0; i < N; i++) {
		scanf("%lf", &a[i]);
	}
	for(int i = 0; i < N; i++) {
		scanf("%lf", &b[i]);
	}
	cudaMemcpy(dev_a, a, sizeof(double) * N, cudaMemcpyHostToDevice);
	cudaMemcpy(dev_b, b, sizeof(double) * N, cudaMemcpyHostToDevice);
	dif<<<1024, 1024>>>(dev_a, dev_b, dev_c, N);
	cudaMemcpy(c, dev_c, sizeof(double) * N, cudaMemcpyDeviceToHost);
	for(int i = 0; i < N; i++) {
		printf("%10.10lf ", c[i]);
	}
	cudaFree(dev_c);
	cudaFree(dev_b);
	cudaFree(dev_a);
	free(a);
	free(b);
	free(c);
	return 0;
}
```

Результаты

N	cpu	1,32	32,64	128,128	256,256	1024,1024
100	0.000000	0.000000	0.000000	0.000000	0.000000	0.015625
1000	0.000000	0.015625	0.015625	0.000000	0.000000	0.000000
10000	0.015625	0.031250	0.031250	0.015625	0.015625	0.015625
100000	0.171875	0.125000	0.078125	0.156250	0.156250	0.125000
1000000	1.781250	1.187500	1.312500	1.218750	1.187500	0.125000

10000000	17.671875	12.375000	11.750000	11.250000	13.093750	10.968750

Выводы

Алгоритм программы максимально прост. Поскольку основной смысл первой лабораторной работы реализация примитивных операций, выполняемых на видеокарте. Теперь немного проанализируем полученные результаты, как видно из заполненных значений при N приближенном к реальной жизни программа написанная на gpu начинает превосходить программу написанную на cpu. Но при небольших N, cpu все же опережает gpu, это связано с времязатратным созданием потоков.