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: Е.С. Пищик
Группа: 8О-406Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2022

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 8. Реверс вектора.

Программное и аппаратное обеспечение

GPU via SSH:

- Compute capability: 2.1
- Name: GeForce GT 545
- Total Global Memory: 3150381056
- Shared memory per block: 49152
- Registers per block: 32768
- Warp size: 32
- Max threads per block: (1024, 1024, 64)
- Max block: (65535, 65535, 65535)
- Total constant memory: 65536
- Multiprocessors count: 3

CPU via SSH: Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz
RAM via SSH: 16 Gb.
HDD via SSH: 471 Gb.
OS: Windows 10
OS via SSH: Ubuntu 16.04.6 LTS
IDE: Visual Studio Code
Compiler via SSH: nvcc

Метод решения

Для реверса вектора нужно выделить память под два вектора, один будет исходным, туда считаются данные, а дальше в ядре в цикле будем записывать из входного вектора элемент с индексом n - i - 1 в выходной вектор по индексу i.

Описание программы

Программа состоит из одного основного файла gpu.cu. Файл состоит из функций scan, random, init, kernel, main. scan - функция читает данные из потока ввода в вектор. random - заполняет вектор рандомными числами. init - функция, которая заполняет вектор по одной из двух предложенных стратегий, либо используется функция scan, либо random (для варианта работающего на чекере в функции main нужно установить значение переменной random flag равной false, random flag = true нужен для различных тестов, когда random_flag = true, переменную n нужно приравнуть размеру вектора для теста, например int n = 100000). main - основная функция программы, здесь происходит считываение входных данных, выделение памяти под вектора на host устройстве, выделение памяти под вектора на device устройстве (при помощи функций cudaMalloc),

копирование данных с host на device (при помощи cudaMemcpy). При помощи функций cudaEventCreate, cudaEventRecord, cudaEventSynchronize происходит измерение времени работы алгоритма. После всех операций по выделению памяти, копированию векторов, созданию переменных для подсчета времени запускается функция kernel на device устройстве. После выполнения алгоритма идет вывод в потом ошибок времени затраченного на алгоритм, вывод в потом вывода реверснутого вектора, очищение памяти на host устройстве и device устройстве (при помощи cudaFree).

В функции ядра, мы просто вычитаетм абсолютный номер потока - переменная ind. А также количество всех наших потоков - переменная step. Цикл нам нужен для того, чтобы если элементов больше чем потоков, потоки которые обработали уже свои элементы по индексу i, начинали обрабатывать элементы i + step, при этом мы ограничиваем цикл числом n - размер вектора, чтобы не выйти за границу. Функция ядра вызывается как kernel<<<количество блоков, количество потоков в блоке>>>. init_vct - это наш исходный вектор, res_vct - реверснутый вектор, n - число элементов в векторе.

```
__global__ void kernel(double *init_vct, double *res_vct, int n) {
	int ind = blockDim.x * blockIdx.x + threadIdx.x;
	int step = blockDim.x * gridDim.x;

	for(int i = ind; i < n; i += step) {
		res_vct[i] = init_vct[n - i - 1];
	}
}
```

Результаты

CPU

Число n, ед	10^5	10^6	10^7	$5 * 10^7$	10^8
Время, ms	0.772	5.612	60.232	209.346	393.256

GPU kernel<<<1,32>>

Число n, ед	10^5	10^6	10^7	$5 * 10^7$	10^8
Время, ms	1.859	18.492	184.724	923.489	1846.197

GPU kernel<<<32,32>>

Число n, ед	10^5	10^6	10^7	$5 * 10^7$	10^8
Время, ms	0.130	1.212	12.047	60.135	120.671

GPU kernel<<<128,128>>

Число n, ед	10^5	10^6	10^7	$5 * 10^7$	10^8
Время, ms	0.056	0.484	4.796	23.984	48.146

GPU kernel<<<512,512>>

Число n, ед	10^5	10^6	10^7	$5 * 10^7$	10^8
Время, ms	0.067	0.467	4.700	23.457	46.873

GPU kernel<<<1024,1024>>

Число n, ед	10^5	10^6	10^7	$5 * 10^7$	10^8
Время, ms	0.180	0.561	4.635	23.247	46.588

Выводы

Я научился работать с базовыми основами технологии CUDA. Узнал как взаимодействует CPU и GPU устройства, как устроены потоки в технологии CUDA. Применил полученные знания на задаче параллельного реверса вектора. Сравнил результаты полученные на CPU, с результатами на GPU при различных размерах тестов и различных конфигурациях ядер. Сделал выводы, что иногда CPU может работать даже быстрее чем GPU, например на каких-то небольших тестах или если выбрана неудачная комбинация числа потоков и числа блоков. Однако на больших тестах GPU очень сильно превосходит результат полученный на CPU почти на всех конфигурациях ядра. При этом начиная с некоторых конфигураций ядер, если брать большее число блоков и потоков, то результат почти не будет улучшаться, относительно ядер с меньшим числом блоков и потоков.