МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Факультет прикладной математики и физики
Кафедра вычислительной математики и программирования

Лабораторная работа №1
по курсу «Программирование графических процессоров»

Изучение технологии CUDA

Выполнил: Иларионов Д.А.
Группа: М8О-408Б-17
Преподаватели: Крашенинников К.Г.,
Морозов А.Ю.

Москва, 2020

Условие

Цель работы: Знакомство и установка ПО для работы с параллельными процессами и работой программ на GPU (CUDA). Реализация одной из простых операций над векторами в памяти графического процессора.

Вариант: 4. Поэлементное нахождение минимума векторов.

Программное и аппаратное обеспечение

GPU:

- Name: GeForce GTX 1060
- Compute capability: 6.1
- Частота видеопроцессора: 1404 – 1670 (Boost) МГц
- Частота памяти: 8000 МГц
- Графическая память: 6144 МБ
- Разделяемая память: отсутствует
- Количество регистров на блок: 65536
- Максимальное количество блоков: (2147483647, 65535, 65535)
- Максимальное количество нитей: (1024, 1024, 64)
- Количество мультипроцессоров: 10

Сведения о системе:

- Процессор: Intel Core i7-8750H 2.20GHz x 6
- Оперативная память: 16 ГБ
- SSD: 128 ГБ
- HDD: 1000 ГБ

Программное обеспечение:

- OS: Windows 10
- IDE: Visual Studio 2019
- Компилятор: nvcc

Метод решения

Заполняем оба массива. На каждый поток идет 1 элемент с каждого массива с одинаковым индексом. Идет сравнение и выбирается тот, который меньше. Данное число записывается в третий массив. Если элементов больше, чем потоков, то поток обрабатывает сразу несколько элементов, которые находятся на расстоянии = grimDim*blockDim. Если полученное id потока больше, чем номер последнего элемента массивов, то данный элемент не обрабатывается. Поэтому, при большом количестве потоков и время выполнение программы дольше даже на небольших данных.

Описание программы

Файл kernel.cu

```cpp
#include "cuda_runtime.h"
#include "device_launch_parameters.h"

#include <stdio.h>
#include <iostream>
#include <fstream>
#include <string>
#include <algorithm>

using namespace std;

#define FIND_ERR(call) { gpuAssert((call), __FILE__, __LINE__); }

inline void gpuAssert(cudaError_t cudaStatus, const char* file, int line, bool abort =
true)
{
	if (cudaStatus != cudaSuccess)
	{
		fprintf(stderr, "ERROR: CUDA failed in %s:%d: %s\n", file, line,
cudaGetErrorString(cudaStatus));
		exit(0);
	}
}

__global__ void findMin(int N, double* v1, double* v2) {
	int tId = threadIdx.x + blockIdx.x * blockDim.x;
	while (tId < N) {
		if (v1[tId] < v2[tId]) {
			v2[tId] = v1[tId];
		}
		tId += blockDim.x * gridDim.x;
	}
}

//cpu
void findMinCPU(int N, double* v1, double* v2) {
	for (int i = 0 ; i < N ; ++i) {
		v2[i] = min(v1[i], v2[i]);
	}
}

int main()
{
	std::ios::sync_with_stdio(false);
	int N;
	cin >> N;
	if (N <= 0) {
		return 0;
	}

	double* vector1 = new double[N];
	double* vector2 = new double[N];

	double* fastV1, *fastV2;


	// Инициализируем память на GPU
	FIND_ERR(cudaMalloc(&fastV1, N * sizeof(double)));
	FIND_ERR(cudaMalloc(&fastV2, N * sizeof(double)));

	for (int i = 0; i < N; i++) {
		cin >> vector1[i];
	}
```

```
	for (int i = 0; i < N; i++) {
		cin >> vector2[i];
	}


	//CPU FUNCT
	//findMinCPU(N, vector1, vector2, result);



	//Копируем в память GPU
	FIND_ERR(cudaMemcpy(fastV1, vector1, N * sizeof(double),
cudaMemcpyHostToDevice));
	FIND_ERR(cudaMemcpy(fastV2, vector2, N * sizeof(double),
cudaMemcpyHostToDevice));

	//запись времени
	/*
	cudaEvent_t start, end;
	FIND_ERR(cudaEventCreate(&start));
	FIND_ERR(cudaEventCreate(&end));
	FIND_ERR(cudaEventRecord(start));
	*/

	findMin <<<256, 512>>> (N, fastV1, fastV2); //функция
	FIND_ERR(cudaGetLastError()); //просмотр ошибок
	/*
	FIND_ERR(cudaEventRecord(end));
	FIND_ERR(cudaEventSynchronize(end));
	float t;
	FIND_ERR(cudaEventElapsedTime(&t, start, end));
	FIND_ERR(cudaEventDestroy(start));
	FIND_ERR(cudaEventDestroy(end));*/
	//конец записи

	//printf("time = %f\n", t);

	// Копируем результат с ГПУ
	FIND_ERR(cudaMemcpy(vector2, fastV2, N * sizeof(double),
cudaMemcpyDeviceToHost));

	for (int i = 0; i < N; i++) {
		printf("%.10e ", vector2[i]);
	}

	// Освобождение памяти
	FIND_ERR(cudaFree(fastV1));
	FIND_ERR(cudaFree(fastV2));


	delete[] vector1;
	delete[] vector2;


    return 0;
}
```

Результаты

Таблица работы программы с разными конфигурациями ядра GPU и с CPU (в миллисекундах)

L / Core [ms]	(8, 8)	(16, 16)	(32, 32)	(64, 64)	(128, 128)	(256, 256)	(512, 512)	(1024, 1024)	CPU
8	0.008192	0.008192	0.009216	0.008192	0.008192	0.011264	0.028672	0.102400	0.008000
64	0.007168	0.008192	0.008192	0.008192	0.008192	0.011264	0.029696	0.102400	0.010500
512	0.018432	0.009216	0.008160	0.008192	0.008192	0.011264	0.029696	0.102400	0.020200
4096	0.107168	0.030720	0.012000	0.008192	0.008832	0.012160	0.030368	0.102400	0.078500
32768	0.817248	0.209408	0.057312	0.018432	0.013280	0.016192	0.035200	0.110176	0.707600
262144	8.078752	1.914880	0.489024	0.130400	0.054880	0.058144	0.071488	0.148490	5.835600
2097152	62.24589	16.11206	3.873280	1.012096	0.376544	0.390144	0.379904	0.457728	38.87759
16777216	383.3378	102.9375	20.89021	5.671872	2.555904	2.673664	2.425856	2.387840	318.2690
100 Mil	1885.363	505.9977	135.9516	37.21626	14.10048	15.50774	14.23112	9.977856	1990.524

Выводы

Данный алгоритм очень прост, мне не составило особого труда написать программу. Но данная ЛР заключается в другом – познакомиться с работой графического процессора, памятью видеокарты, основами работы в CUDA. Данная задача не сильно подходит для реализации ее на видеопроцессоре, потому что даже при 2 мил. данных не очень много и алгоритм быстрее реализуется на CPU. При работе с GPU тратится лишь дополнительно время на выделение потоков, поэтому и время работы больше в целом. Были глупые ошибки, из-за которых лаба долго не проходила на чекер. Например, в потоковой функции я вместо while почему-то написал int, в итоге, обрабатывались не все блоки. Еще пришлось отключить синхронизацию данных при вводе (std::ios::sync_with_stdio(false)), без этой строчки программа не проходила по времени, данный совет дал мне одногруппник. А в целом лаба не показалась мне особо сложной. Очень много времени ушло на замер тестов.