Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Сибирский государственный университет
телекоммуникаций и информатики» (СибГУТИ)

09.03.01 "Информатика и вычислительная техника"
профиль "Программное обеспечение средств
вычислительной техники и автоматизированных систем"

ОТЧЕТ
по дисциплине «Программирование графических процессоров»
лабораторная работа 1

Выполнил:
Разумов Д. Б.
студент группы ИП-811

Проверил:
преподаватель Малков Е.А.

Новосибирск 2020

Оглавление

Задание

Цель: освоить студентами удаленную консоль сервера с установленным аппаратным и программным обеспечением, необходимым для освоения курса, или/и установить необходимое ПО на собственные компьютеры. Познакомиться с основными конструкциями программного интерфейса CUDA.

Лабораторная: написать программу для сложения двух векторов, выполняемую на GPU (использовать программные конструкции из примера в первой лекции). Построить графики зависимости времени вычисления от размерности векторов N в диапазоне от 1<<10 до 1<<23, при различных конфигурациях нитей. Оформить результаты в форме отчета (электронные документы, без распечатывания).

Цель: приобрести студентами «априорный опыт», до изучения требований, предъявляемых к оптимальной конфигурации нитей, в разработке кода, исполняемого на GPU. Дать студентам почувствовать необходимость в инструментарии для профилирования CUDA-программ.

Листинг

```
#include <stdio.h>
#include <time.h>
#include <malloc.h>
const int N = 1 << 20;

__global__ void gInitVectors(double* vector1, double* vector2) {
	for (int i = 0; i < N; i++) {
		vector1[i] = (double)i; //rand();
		vector2[i] = (double)i;
	}
}

__global__ void gVectorAddition(double* vector1, double* vector2, double* vectorSum, int threads_cnt) {
	int i = threadIdx.x + blockIdx.x * blockDim.x;
	if (i >= N)
		return;
	vectorSum[i] = vector1[i] + vector2[i];

}

double testingThreadsOfDevice(int threads_cnt/*, double* vector1_d, double* vector2_d*/) {
	double *vectorSum_d, *vectorSum_h;
	vectorSum_h = (double*) calloc(N, sizeof(double));
	cudaMalloc((void**)&vectorSum_d, N * sizeof(double));
```

```
	double *vector1_d, *vector2_d;
	cudaMalloc((void**)&vector1_d, N * sizeof(double));
	cudaMalloc((void**)&vector2_d, N * sizeof(double));
	gInitVectors <<< 1, 1 >>> (vector1_d, vector2_d);

	/// проверка: ///
	/*cudaMemcpy(vectorSum_h, vector1_d, N * sizeof(double),
cudaMemcpyDeviceToHost);
	for (int i = 0; i < N; i++)
		fprintf(stderr, "%g ", vectorSum_h[i]);
	printf("\n");
	*/

	struct timespec mt1, mt2;
	clock_gettime(CLOCK_REALTIME, &mt1);

	gVectorAddition <<< N / threads_cnt, threads_cnt >>>
		(vector1_d, vector2_d, vectorSum_d, threads_cnt); //запуск фу-ии на
GPU
	cudaDeviceSynchronize(); //синхронизация потоков

	/// проверка: ///
	/*cudaMemcpy(vectorSum_h, vectorSum_d, N * sizeof(double),
cudaMemcpyDeviceToHost);
	for (int i = 0; i < N; i++)
		fprintf(stderr, "%g ", vectorSum_h[i]);
	printf("\n");
	*/
```

```
	clock_gettime(CLOCK_REALTIME, &mt2);
	double seconds_double = (double)(mt2.tv_sec - mt1.tv_sec) +
	(double)(mt2.tv_nsec - mt1.tv_nsec) / 1e9; ///время в секундах
	printf("blocks = %d, threads per block = %d seconds = %e \n",
		N / threads_cnt, threads_cnt, seconds_double);

	cudaFree(vector1_d);
	cudaFree(vector2_d);
	cudaFree(vectorSum_d);
	free(vectorSum_h);
	return seconds_double;
}

int main() {
	//открытие файла для записи результатов:
	FILE *fout;
	if ((fout = fopen("output.csv", "w")) == NULL) {
		printf("error: can't open output.csv \n");
		return 1;
	}
	//инициализация векторов:

	//тестирование и запись в файл:
	fprintf(fout, "threads_per_block;time_in_seconds;\n");
	for (int i = 1; i <= 1024; i *= 2)//i < 100 ? i *= 2 : i += 50)
		fprintf(fout, "%d;%e;\n", i, testingThreadsOfDevice(i/*, vector1_d,
vector2_d*/));
```

```
    return 0;
}
```

Скриншоты

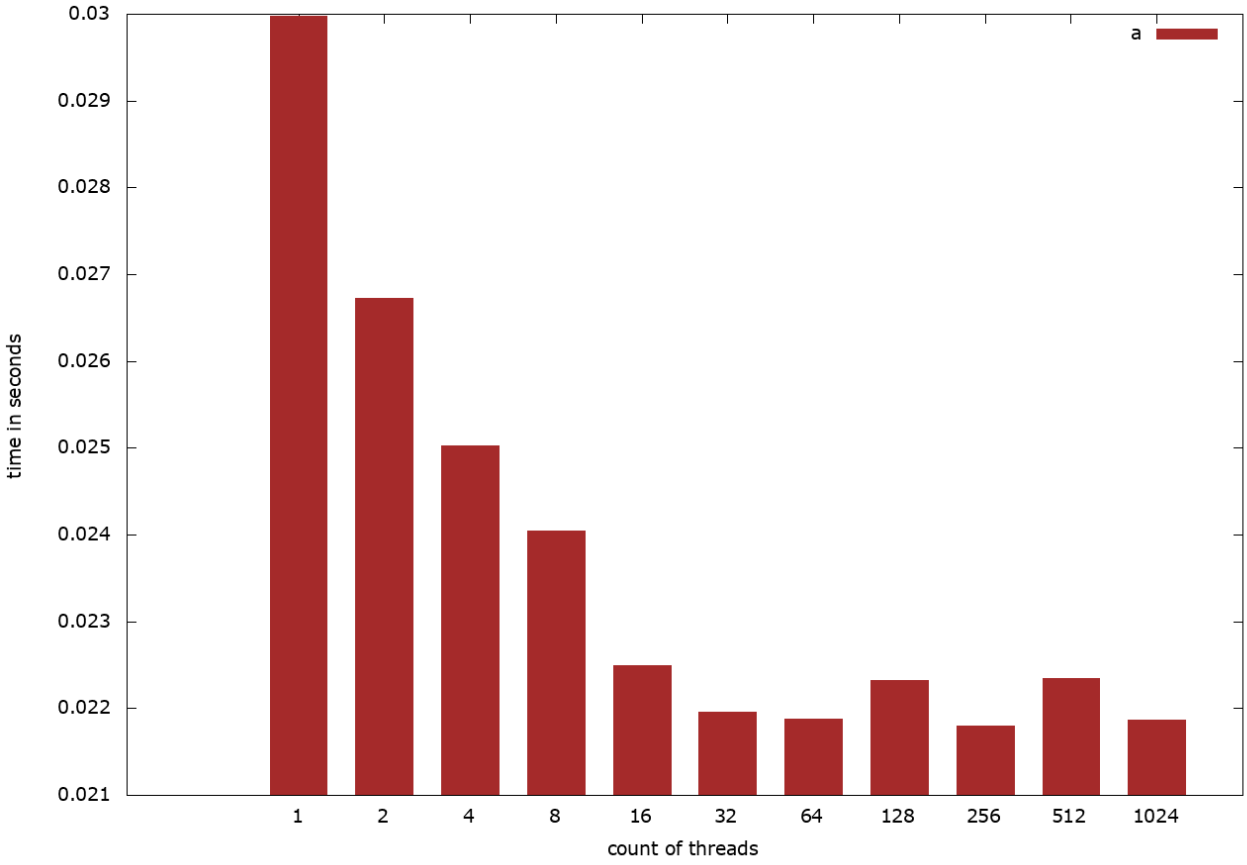

Рисунок 1 — гистограмма, где показано время выполнения сложения векторов в зависимости от количества потоков

Заключение

Я познакомился с CUDA API и проверил экспериментально зависимость скорости работы моей видеокарты (NVIDIA Geforce MX350) от различных конфигураций потоков.