МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Кафедра вычислительной математики и программирования

спецкурс «Параллельные и распределенные вычисления»

ОТЧЕТ

Лабораторная работа № 1
«Освоение программного обеспечения среды программирования NVIDIA»

Выполнила: Алексеева М.А.
Группа: М80-114М-22
Преподаватель: Семенов С. А.

Москва, 2022

Содержание

1. Постановка задачи

Вариант 1 - вычислить функцию корня из аргумента.

2. Описание решения

Задание будет исполняться с помощью встроенной функции sqrt. Создаем массив и инициализируется каждый элемент числом i, где i - индекс элемента в массиве. Далее для каждого элемента массива вычисляется корень.

3. Аппаратное обеспечение и ПО

Процессор Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80 GHz
Оперативная память 8,00 ГБ
Тип системы 64-разрядная операционная система, процессор x64
Ядер : 4
Виртуализация: Включена

Tesla K80, 11441 MiB
Количество потоковых процессоров:
2496из 10752 (GA102)
Частота ядра:
562 МГциз 2233 (Playstation 5 GPU)

Частота в режиме Boost:
824 МГциз 2903 (Radeon Pro W6600)
Количество транзисторов:
7,100 млниз 14400 (GeForce GTX 1080 SLI (мобильная))

Программное обеспечение:
Google Collab

4. Основные моменты кода

```
__global__ void SKernel(float *a, float *b, int n) {
    int idx = threadIdx.x + blockIdx.x * blockDim.x;
    int offset = blockDim.x * gridDim.x;
    while(idx < n) {
        b[idx] = sqrt(a[idx]);
        idx += offset;
    }
}
```

5. Результат работы программы

Для GPU	Для CPU
N = 8192	N = 8192
t = 1 с	2 с

```
user200@server-i72:~$ nvcc lab_1.cu
user200@server-i72:~$ ./a.out
0.000000
1.000000
1.414214
1.732051
2.000000
2.236068
2.449490
2.645751
2.828427
3.000000
3.162278
3.316625
3.464102
3.605551
3.741657
3.872983
4.000000
4.123106
4.242640
4.358899
4.472136
4.582576
4.690416
4.795832
4.898980
5.000000
5.099020
5.196152
5.291502
5.385165
5.477226
5.567764
5.656854
5.744563
5.830952
5.916080
6.000000
6.082763
6.164414
6.244998
6.324555
6.403124
6.480741
6.557438
6.633250
6.708204
6.782330
6.855655
6.928203
7.000000
7.071068
7.141428
7.211102
7.280110
7.348469
7.416198
7.483315
7.549834
7.615773
7.681146
7.745967
7.810250
7.874008
7.937254
8.000000
8.062258
8.124039
```

6. Сравнение скорости выполнения на CPU и GPU

При запуске программы с различными значениями N видно, что вычисления на видеокарте произвелись быстрее, чем на процессоре компьютера, при $N>$ 8192.

Время выполнения программы при различных значениях N:

N	GPU время выполнения, мс	CPU время выполнения, мс	t_{CPU}/t_{GPU}
4	1000	1000	1
8	1000	1000	1
16	1000	1000	1
32	1000	1000	1
64	1000	1000	1
128	1000	1000	1
256	1000	1000	1
1024	1000	1000	1
8192	1000	2000	2
32768	4000	5000	1,25
65536	11000	12000	1,0909
131072	25000	27000	1,08
262144	40000	56000	1,4
524288	80000	120000	1,5
1048576	240000	360000	1,5

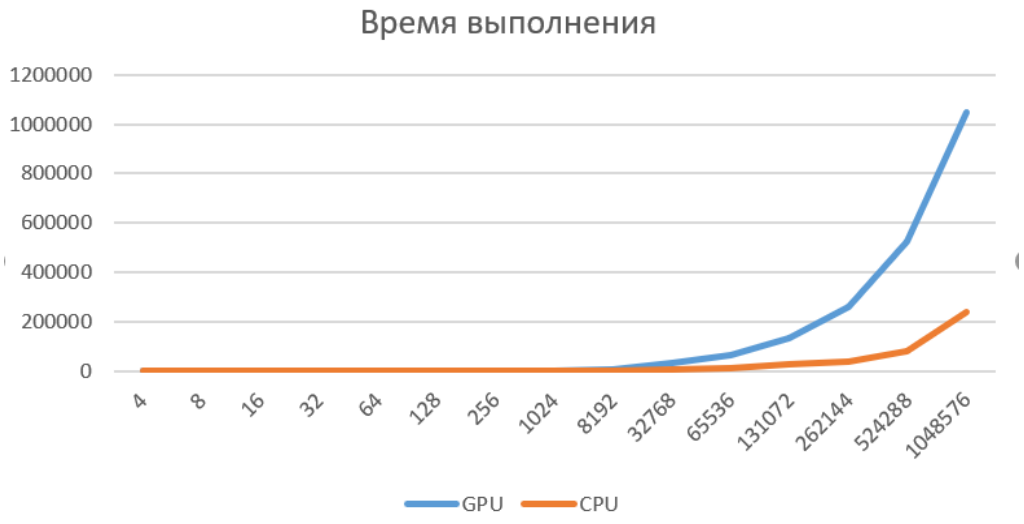

Рис.1 График зависимости времени выполнения программы.

7. Выводы

В Лабораторной работе №1 проведен анализ работы различных программ по решению задачи вычисления функции косинуса.

При сравнении реализации задачи на CPU и GPU, скорость выполнения до N = 8192 существенно не изменяется. До данного числа разницы не замечено. С увеличением числа N разрыв становится существенным, но не более чем в полтора раза.

8. Приложения

Ссылка на Github:

Код программы на GPU:

```
#include <stdlib.h>
#include <stdio.h>

__global__ void SKernel(float *a, float *b, int n) {
    int idx = threadIdx.x + blockIdx.x * blockDim.x;
    int offset = blockDim.x * gridDim.x;
    while(idx < n) {
        b[idx] = sqrt(a[idx]);
        idx += offset;
    }
}

void Printer(float *a, int n){
    for (int i = 0; i < n; i++){
        printf("%f\n", a[i]);
    }
}

void Assigner(float *a, int n){
    for (int i = 0; i < n; i++){
        a[i] = (float)i;
    }
}

int main() {

    int n = 100;
    int size = n * sizeof(float);

    float *aDev = NULL, *bDev = NULL;
    float *a = NULL, *b = NULL;
```

```
    cudaMalloc((void **) &aDev, size);
    cudaMalloc((void **) &bDev, size);

    a = (float *) malloc(size);
    b = (float *) malloc(size);

    Assigner(a, n);

    cudaMemcpy(aDev, a, size, cudaMemcpyHostToDevice);
    cudaMemcpy(bDev, b, size, cudaMemcpyHostToDevice);

    SKernel<<<256, 256>>> (aDev, bDev, n);

    cudaMemcpy(b, bDev, size, cudaMemcpyDeviceToHost);

    Printer(b, n);

    cudaFree(aDev);
    cudaFree(bDev);

    free(a);
    free(b);
}
```