

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 8

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

Проверил:  
преподаватель Малков Е.А.

## Оглавление

Задание.....	3
Листинг.....	4
Файл task1.cu.....	4
Файл task2.cu.....	11
Скриншоты.....	16
Об используемом GPU.....	17
Заключение.....	18

# Задание

## Задание:

- сравните производительность программ, реализующих `saxpy` на основе библиотек `thrust` и `cuBLAS`.
- выявите периодичность солнечной активности, опираясь на данные наблюдений о числе Вольфа, представленные по адресу:  
[http://www.gaoran.ru/database/csa/daily\\_wolf\\_r.html](http://www.gaoran.ru/database/csa/daily_wolf_r.html)

## Цель:

научиться пользоваться прикладными библиотеками CUDA.

# ЛИСТИНГ

## Файл task1.cu

```
#include <stdio.h>

#include <stdlib.h>

#include <thrust/host_vector.h>

#include <thrust/device_vector.h>

#include <thrust/transform.h>

#include <thrust/fill.h>

#include <thrust/sequence.h>

#include <cuda_runtime.h>

#include "cublas_v2.h"

#define CUDA_CHECK_RETURN(value) {\
    cudaError_t _m_cudaStat = value;\
    if (_m_cudaStat != cudaSuccess) {\
        fprintf(stderr, "Error \"%s\" at line %d in file %s\n",\
            cudaGetErrorString(_m_cudaStat), __LINE__, __FILE__);\
        exit(1);\
    }\
} //макрос для обработки ошибок

#define CUBLAS_CHECK_RETURN(value) {\
    cublasStatus_t stat = value;\
    if (stat != CUBLAS_STATUS_SUCCESS) {\
        fprintf(stderr, "Error at line %d in file %s\n",\
            __LINE__, __FILE__);\
    }\
}
```

```

        exit(1);\
    }\
} //макрос для обработки ошибок

struct saxpy_functor
{
    const float a;
    saxpy_functor(float _a) : a(_a) {}
    __host__ __device__
    float operator()(float x, float y) {
        return a*x+y;
    }
};

void saxpy(float a, thrust::device_vector<float>& x,
    thrust::device_vector<float>& y)
{
    saxpy_functor func(a);
    thrust::transform(x.begin(), x.end(), y.begin(), y.begin(), func);
}

float saxpy_thrust(long int arr_size, cudaEvent_t start, cudaEvent_t stop)
{ //SAXPY с помощью thrust
    /// создание и заполнение векторов векторов:
    thrust::host_vector<float> h1(arr_size);
    thrust::host_vector<float> h2(arr_size);
    thrust::sequence(h1.begin(), h1.end());
    //thrust::fill(h2.begin(), h2.end(), 0.0);

```

```

thrust::device_vector<float> d1 = h1;
thrust::device_vector<float> d2 = h2;

/*
printf("before saxpy:\n");
for (int i=0; i<16; i++) {
    printf("i = %d; h1[i]=%g; h2[i]=%g\n",i, h1[i], h2[i]);
}
*/

cudaEventRecord(start, 0);
    saxpy(2.5, d1, d2);
cudaEventRecord(stop, 0);
cudaEventSynchronize(stop);
float time;
cudaEventElapsedTime(&time, start, stop);

/// вывод содержимого векторов после сложения:
/*
h2 = d2;
h1 = d1;
printf("after saxpy:\n");
for (int i=0; i<16; i++) {
    printf("i = %d;\t h1[i] = %g;\t h2[i] = %g\n",i, h1[i], h2[i]);
}
*/

```

```

        return time;
    }

float saxpy_cublas(long int arr_size, cudaEvent_t start, cudaEvent_t stop)
{ //SAXPY с помощью cublas:
    const size_t size_in_bytes = (arr_size * sizeof(float));

    float *A_dev;
    cudaMalloc( (void **) &A_dev, size_in_bytes );
    float *B_dev;
    cudaMalloc( (void **) &B_dev, size_in_bytes );
    float *A_hos;
    cudaMallocHost( (void **) &A_hos, size_in_bytes );
    float *B_hos;
    cudaMallocHost( (void **) &B_hos, size_in_bytes );
    memset(A_hos, 0, size_in_bytes);
    memset(B_hos, 0, size_in_bytes);

    //инициализация библиотеки CUBLAS
    cublasHandle_t cublas_handle;
    CUBLAS_CHECK_RETURN(cublasCreate(&cublas_handle));

    //заполнение массива A:
    for (int i=0; i < arr_size; i++){
        A_hos[i] = (float)i;
    }
    /*

```

```
printf("before saxpy (cublas):\n");
for (int i = 0; i < 16; i++) {
    printf("i = %d;\t h1[i] = %g;\t h2[i] = %g\n", i+1, A_hos[i], B_hos[i]);
}*/
```

```
const int num_rows = arr_size; //arr_size
const int num_cols = 1; //1
const size_t elem_size = sizeof(float);
```

```
//Копирование матрицы с числом строк arr_size и одним столбцом с
//хоста на устройство
cublasSetMatrix(num_rows, num_cols, elem_size, A_hos,
    num_rows, A_dev, num_rows); //leading dimension
```

```
//Очищаем массив на устройстве
cudaMemset(B_dev, 0, size_in_bytes);
```

```
// выполнение SingleAlphaXPlusY (saxpy)
const int stride = 1; //шаг (каждый stride элемент берется из массива)
float alpha = 2.5F;
```

```
cudaEventRecord(start, 0);
    cublasSaxpy(cublas_handle, arr_size, &alpha, A_dev,
        stride, B_dev, stride);
cudaEventRecord(stop, 0);
cudaEventSynchronize(stop);
```



```

float time;
cudaEventElapsedTime(&time, start, stop);

/*
//Копирование матриц с числом строк arr_size и одним столбцом с
//устройства на хост
cublasGetMatrix(num_rows, num_cols, elem_size, A_dev,
                num_rows, A_hos, num_rows);
cublasGetMatrix(num_rows, num_cols, elem_size, B_dev,
                num_rows, B_hos, num_rows);
printf("after saxpy (cublas):\n");
for (int i = 0; i < 16; i++) {
    printf("i = %d;\t h1[i] = %g;\t h2[i] = %g\n", i+1, A_hos[i], B_hos[i]);
}
*/

// Освобождаем ресурсы на устройстве
cublasDestroy(cublas_handle);
cudaFree(A_dev);
cudaFree(B_dev);
// Освобождаем ресурсы на хосте
cudaFreeHost(A_hos);
cudaFreeHost(B_hos);
//сброс устройства, подготовка для выполнения новых программ
//cudaDeviceReset();
return time;
}

```

```

int main(){
    /// информация об используемом устройстве:
    cudaDeviceProp deviceProp;
    cudaGetDeviceProperties(&deviceProp, 0);
    printf("device: %s \n\n", deviceProp.name);

    cudaEvent_t start, stop;
    cudaEventCreate(&start);
    cudaEventCreate(&stop);

    ///размерность массивов:
    long int arr_size = 1 << 20, pow = 20;
    for ( ; pow <= 25; pow++, arr_size = 1 << pow) {
        printf("arr_size = 1 << %ld\n", pow);
        float time = saxpy_thrust(arr_size, start, stop);
        printf("Thrust time = %f ms\n", time);
        time = saxpy_cublas(arr_size, start, stop);
        printf("CuBLAS time = %f ms\n\n", time);
    }

    cudaEventDestroy(start);
    cudaEventDestroy(stop);
    return 0;
}

```

## Файл task2.cu

```
#include <malloc.h>

#include <string.h>

#include <stdio.h>

#include <vector>

#include <fstream>

#include <iostream>

using namespace std;

#include <cuFFT.h>

#define BATCH 1 //размер данных для обработки

#define CUDA_CHECK_RETURN(value) {\
    cudaError_t _m_cudaStat = value;\
    if (_m_cudaStat != cudaSuccess) {\
        fprintf(stderr, "Error \"%s\" at line %d in file %s\n",\
            cudaGetErrorString(_m_cudaStat), __LINE__, __FILE__);\
        exit(1);\
    }\
} //макрос для обработки ошибок

#define CUFFT_CHECK_RETURN(value) {\
    cufftResult stat = value;\
    if (stat != CUFFT_SUCCESS) {\
        fprintf(stderr, "Error at line %d in file %s\n",\
            __LINE__, __FILE__);\
        exit(1);\
    }\
}
```

```

    }\
} //макрос для обработки ошибок

int main(void) {
    string line, buffer;

    cufftHandle plan; //дескриптор плана конфигурации (нужен для
оптимизации под выбранную аппаратуру)

    cufftComplex *hos_data, *dev_data; //массивы для комплексных чисел
    vector<string> file_string; //одна строка в файле = массив из 4 чисел в виде
string
    vector<float> Wolf_nums; //числа Вольфа (весь диск)
    vector<float> freq; //массив частот после преобразования Фурье (на
каждый день одно число)
    vector<float> power; //массив значений  $\wedge 2$  из преобразования Фурье
    ifstream in;

    for (int i = 1938; i <= 1991; i++) { //цикл по годам
        //открыть файл i-го года
        string path = string("data/w") + to_string(i) + string(".dat");
        in.open(path);
        if (!in.is_open()) {
            fprintf(stderr, "can't open a file data/w%d.dat\n", i);
            return -1;
        }

        while(getline(in, line)) { //считать строку из файла
            buffer = "";
            line += " "; //добавить пробел для обработки последнего числа

```

```

for (int k = 0; k < line.size(); k++) {
    if (line[k] != ' ') {
        buffer += line[k]; //записать число посимвольно в
buffer
    }
    else { //если пробел, то есть получено число в виде
строки
        if (buffer != "")
            file_string.push_back(buffer);
            buffer = "";
        }
    }

    if (file_string.size() != 0) {
        if (file_string[2] == "999") { //если число Вульфа
неизвестно,
            file_string[2] = to_string(Wolf_nums.back()); //то
взять предыдущее значение
        }
        Wolf_nums.push_back(stoi(file_string[2]));
//преобразовать строку в число, добавить в массив чисел
        file_string.clear(); //очистить строку
    }
} //end of while(getline(in, line))

in.close();
} //end of for(int i = 1938; i <= 1991; i++)

int N = Wolf_nums.size();

```

```

cudaMalloc((void**)&dev_data, sizeof(cufftComplex) * N * BATCH);
hos_data = new cufftComplex[N * BATCH];
for (int i = 0; i < N * BATCH; i++) {
    hos_data[i].x = Wolf_nums[i]; //действительная часть
    hos_data[i].y = 0.0f; //мнимая часть
}

cudaMemcpy(dev_data, hos_data, sizeof(cufftComplex) * N * BATCH,
cudaMemcpyHostToDevice);

//создание плана, преобразование Фурье происходит из комплексных
чисел в комплексные:
CUFFT_CHECK_RETURN(cufftPlan1d(&plan, N * BATCH, CUFFT_C2C,
BATCH));

//совершить быстрое преобразование Фурье (FFT):
CUFFT_CHECK_RETURN(cufftExecC2C(plan, dev_data, dev_data,
CUFFT_FORWARD));

//синхронизация:
CUDA_CHECK_RETURN(cudaDeviceSynchronize());

//копируем обратно на хост то, что получено после FFT:
CUDA_CHECK_RETURN(cudaMemcpy(hos_data, dev_data, N *
sizeof(cufftComplex), cudaMemcpyDeviceToHost));

power.resize(N / 2 + 1);
for (int i = 1; i <= N / 2; i++) {
    //преобразовать значения, т.к. комплексные числа тяжело
интерпретировать:
    power[i] = sqrt(hos_data[i].x * hos_data[i].x + hos_data[i].y *
hos_data[i].y);

```

```
}
```

```
float max_freq = 0.5; //максимальная частота
```

```
freq.resize(N / 2 + 1);
```

```
for (int i = 1; i <= N / 2; i++) {
```

```
    //получаем равномерно распределенную сетку частот:
```

```
    freq[i] = 1 / (float(i) / float(N/2) * max_freq);
```

```
}
```

```
int maxind = 1; //найти максимальное значение
```

```
for (int i = 1 ; i <= N / 2; i++) {
```

```
    if (power[i] > power[maxind])
```

```
        maxind = i;
```

```
}
```

```
//freq[maxind] - это количество дней при максимальной частоте?
```

```
printf("calculated periodicity = %f years\n", freq[maxind] / 365);
```

```
cufftDestroy(plan);
```

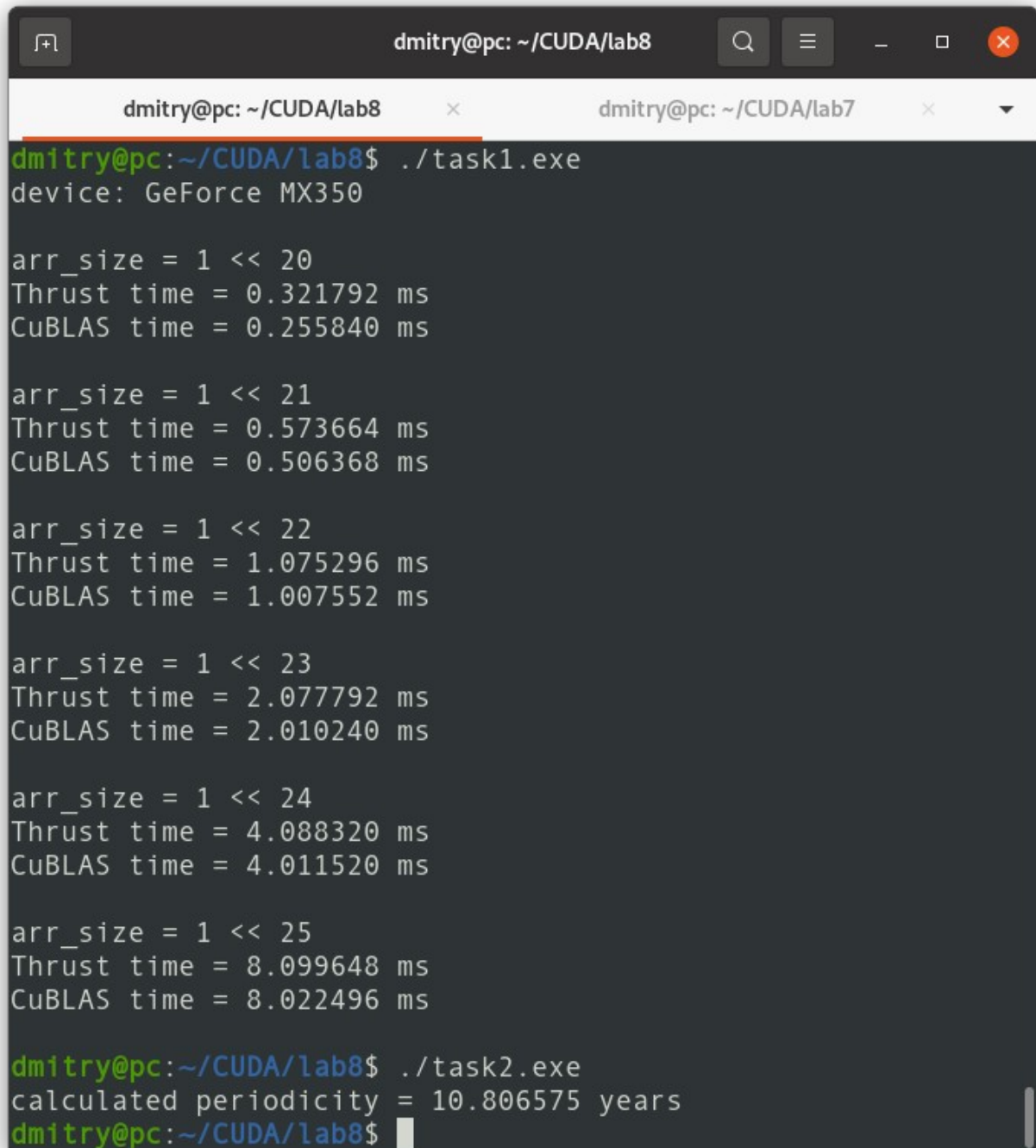
```
cudaFree(dev_data);
```

```
free(hos_data);
```

```
return 0;
```

```
}
```

# Скриншоты



```
dmitry@pc: ~/CUDA/lab8
dmitry@pc: ~/CUDA/lab8$ ./task1.exe
device: GeForce MX350

arr_size = 1 << 20
Thrust time = 0.321792 ms
CuBLAS time = 0.255840 ms

arr_size = 1 << 21
Thrust time = 0.573664 ms
CuBLAS time = 0.506368 ms

arr_size = 1 << 22
Thrust time = 1.075296 ms
CuBLAS time = 1.007552 ms

arr_size = 1 << 23
Thrust time = 2.077792 ms
CuBLAS time = 2.010240 ms

arr_size = 1 << 24
Thrust time = 4.088320 ms
CuBLAS time = 4.011520 ms

arr_size = 1 << 25
Thrust time = 8.099648 ms
CuBLAS time = 8.022496 ms

dmitry@pc:~/CUDA/lab8$ ./task2.exe
calculated periodicity = 10.806575 years
dmitry@pc:~/CUDA/lab8$
```

Рисунок 1 — вывод результатов (обоих программ)



## Об используемом GPU

Device name: GeForce MX350

Global memory available on device: 2099904512 (2002 MByte)

Shared memory available per block: 49152

Count of 32-bit registers available per block: 65536

Warp size in threads: 32

Maximum pitch in bytes allowed by memory copies: 2147483647

Maximum number of threads per block: 1024

Maximum size of each dimension of a block[0]: 1024

Maximum size of each dimension of a block[1]: 1024

Maximum size of each dimension of a block[2]: 64

Maximum size of each dimension of a grid[0]: 2147483647

Maximum size of each dimension of a grid[1]: 65535

Maximum size of each dimension of a grid[2]: 65535

Clock frequency in kilohertz: 1468000

totalConstMem: 65536

Major compute capability: 6

Minor compute capability: 1

Number of multiprocessors on device: 5

Count of cores: 640

Id current device: 0

Id nearest device to 1.3: 0

## Заключение

Код для реализации SAXPY, написанный на Thrust и CuBLAS, имеет почти одинаковую производительность.

Вычисленная мною периодичность солнечной активности равна 10.8, что довольно близко к реальному значению.