

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 6

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

Проверил:
преподаватель Малков Е.А.

Оглавление

Задание.....	3
Листинг.....	4
Файл lab6.cu.....	4
Файл task1.h.....	5
Файл task1.cu.....	6
Файл task2.h.....	8
Файл task2.cu.....	8
Файл task3.h.....	12
Файл task3.cu.....	12
Скриншоты.....	17
Об используемом GPU.....	18
Заключение.....	19

Задание

Задание:

- разработать и программно реализовать алгоритм для сравнения производительности копирования устройство->хост (и наоборот) данных, размещенных в памяти выделенной на хосте обычным образом и с использованием закрепленных страниц ;
- подобрать оптимальный размер порции данных для реализации сложения векторов с использованием потоков CUDA для распараллеливания копирования и выполнения;
- то же для реализации скалярного умножения.

Цель: изучить преимущества использования потоков CUDA.

ЛИСТИНГ

Файл lab6.cu

```
#include "task1.h"
#include "task2.h"
#include "task3.h"

int main()
{
    cudaDeviceProp prop;
    int whichDevice;

    // проверяем поддерживает ли устройство overlapping computation with
    memory copy
    cudaGetDevice(&whichDevice);
    cudaGetDeviceProperties(&prop, whichDevice);
    if (!prop.deviceOverlap) {
        printf("device will not handle\n");
        return 0;
    }

    //srand(time(0));
    task1();
    task2();
    task3();

    return 0;
}
```

Файл task1.h

```
#ifndef TASK1
#define TASK1

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <cuda.h>
#include "cuda_runtime.h"
#include "device_launch_parameters.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 M 33554432

__global__ void gInitArray(float* arr);
int task1();

#endif
```

Файл task1.cu

```
#include "task1.h"
```

```
__global__ void gInitArray(float* arr) {
    int i = threadIdx.x + blockIdx.x * blockDim.x;
    if (i >= M)
        return;
    arr[i] = (float)i;
}

int task1() {
    //long arr_size = 1 << 20;
    float* host_usual_arr, * host_paged_arr, * dev_arr;

    //выделение обычной памяти на хосте:
    host_usual_arr = (float*)malloc(M * sizeof(float));
    //выделение закрепленной (paged-locked) памяти на хосте:
    CUDA_CHECK_RETURN(cudaHostAlloc((void**)&host_paged_arr,
        M * sizeof(float), cudaHostAllocDefault));
    //выделение памяти на девайсе:
    CUDA_CHECK_RETURN(cudaMalloc((void**)&dev_arr, M * sizeof(float)));

    gInitArray << < M / 128, 128 >> > (dev_arr);
    cudaDeviceSynchronize();

    cudaEvent_t start, stop;
    float elepsedTime;
    cudaEventCreate(&start);
```

```
cudaEventCreate(&stop);  
printf("time of copying arrays:\n");
```

```
cudaEventRecord(start, 0);  
CUDA_CHECK_RETURN(cudaMemcpy(host_paged_arr, dev_arr, M *  
sizeof(float), cudaMemcpyDeviceToHost));  
cudaEventRecord(stop, 0);  
cudaEventSynchronize(stop);  
cudaEventElapsedTime(&elepsedTime, start, stop);  
printf("device array -> host paged array = %f ms\n", elepsedTime);
```

```
cudaEventRecord(start, 0);  
CUDA_CHECK_RETURN(cudaMemcpy(host_usual_arr, dev_arr, M *  
sizeof(float), cudaMemcpyDeviceToHost));  
cudaEventRecord(stop, 0);  
cudaEventSynchronize(stop);  
cudaEventElapsedTime(&elepsedTime, start, stop);  
printf("device array -> host usual array = %f ms\n", elepsedTime);
```

```
cudaEventRecord(start, 0);  
CUDA_CHECK_RETURN(cudaMemcpy(dev_arr, host_paged_arr, M *  
sizeof(float), cudaMemcpyHostToDevice));  
cudaEventRecord(stop, 0);  
cudaEventSynchronize(stop);  
cudaEventElapsedTime(&elepsedTime, start, stop);  
printf("host paged array -> device array = %f ms\n", elepsedTime);
```

```
cudaEventRecord(start, 0);
```

```

    CUDA_CHECK_RETURN(cudaMemcpy(dev_arr, host_usual_arr, M *
sizeof(float), cudaMemcpyHostToDevice));
    cudaEventRecord(stop, 0);
    cudaEventSynchronize(stop);
    cudaEventElapsedTime(&elepsedTime, start, stop);
    printf("host usual array -> device array = %f ms\n", elepsedTime);

    printf("\n");
    free(host_usual_arr);
    cudaFreeHost(host_paged_arr);
    cudaFree(dev_arr);
    return 0;
}

```

Файл task2.h

```

#ifndef TASK2
#define TASK2

#define N (1024*1024)
#define FULL_DATA_SIZE (N*32)

__global__ void add_gpu(int* a, int* b, int* c);
int task2();

#endif

```

Файл task2.cu

```

#include "task1.h"
#include "task2.h"

```



```

__global__ void add_gpu(int* a, int* b, int* c) {

    int tid = threadIdx.x + blockIdx.x * blockDim.x;
    if (tid <= N / 256)
        c[tid] = a[tid] + b[tid];
}

int task2() {
    //cuda-события для замера времени выполнения:
    cudaEvent_t start, stop;
    float elapsedTime;

    cudaStream_t stream0, stream1;
    cudaStreamCreate(&stream0);
    cudaStreamCreate(&stream1);

    printf("\n\tsumming vectors through 2 streams: \n");
    for (long var_size = 1024; var_size <= FULL_DATA_SIZE / 2; var_size *= 2) {
        int* host_a, * host_b, * host_c;

        int* dev_a0, * dev_b0, * dev_c0; // первый gpu input buffer for stream0,
        который будет заполнен случайными числами

        int* dev_a1, * dev_b1, * dev_c1; // второй gpu input buffer for stream1,
        который будет заполнен случайными числами

        //выделение памяти на gpu
        cudaMalloc((void**)&dev_a0, var_size * sizeof(int));
        cudaMalloc((void**)&dev_b0, var_size * sizeof(int));
        cudaMalloc((void**)&dev_c0, var_size * sizeof(int));

        //выделение памяти на gpu

```

```

cudaMalloc((void**)&dev_a1, var_size * sizeof(int));
cudaMalloc((void**)&dev_b1, var_size * sizeof(int));
cudaMalloc((void**)&dev_c1, var_size * sizeof(int));
// выделение page-locked памяти, используемой для стримов
cudaHostAlloc((void**)&host_a, FULL_DATA_SIZE * sizeof(int),
cudaHostAllocDefault);

cudaHostAlloc((void**)&host_b, FULL_DATA_SIZE * sizeof(int),
cudaHostAllocDefault);

cudaHostAlloc((void**)&host_c, FULL_DATA_SIZE * sizeof(int),
cudaHostAllocDefault);

for (int i = 0; i < FULL_DATA_SIZE; i++) {
    host_a[i] = i;//rand();
    host_b[i] = i;// rand();
}

cudaEventCreate(&start);
cudaEventCreate(&stop);
cudaEventRecord(start, 0);

// теперь проитерировать всю дату, через байтные куски:
for (int i = 0; i < FULL_DATA_SIZE; i += var_size * 2) {
    // асинхронно копировать закрытую память на устройство:
    cudaMemcpyAsync(dev_a0, host_a + i, var_size * sizeof(int),
cudaMemcpyHostToDevice, stream0);

    cudaMemcpyAsync(dev_b0, host_b + i, var_size * sizeof(int),
cudaMemcpyHostToDevice, stream0);

    add_gpu <<< var_size / 256, 256, 0, stream0 >>> (dev_a0, dev_b0, dev_c0);

    // копировать дату с устройства на закрытую памятьЖ

```

```

        cudaMemcpyAsync(host_c + i, dev_c0, var_size * sizeof(int),
cudaMemcpyDeviceToHost, stream0);

        // асинхронно копировать закрытую память на устройство
        cudaMemcpyAsync(dev_a1, host_a + i + var_size, var_size * sizeof(int),
cudaMemcpyHostToDevice, stream1);

        cudaMemcpyAsync(dev_b1, host_b + i + var_size, var_size * sizeof(int),
cudaMemcpyHostToDevice, stream1);

        add_gpu <<< var_size / 256, 256, 0, stream1 >>> (dev_a1, dev_b1, dev_c1);

        // копировать дату с устройства на закрытую память
        cudaMemcpyAsync(host_c + i + var_size, dev_c1, var_size * sizeof(int),
cudaMemcpyDeviceToHost, stream1);
    }

    // синхронизируем оба потока
    cudaStreamSynchronize(stream0);
    cudaStreamSynchronize(stream1);
    cudaEventRecord(stop, 0);
    cudaEventSynchronize(stop);
    cudaEventElapsedTime(&elepsedTime, start, stop);
    printf("var_size = FULL_DATA_SIZE / %5ld. Time taken: %f ms\n",
FULL_DATA_SIZE / var_size, elepsedTime);

    //проверка:
    /*
    for (int i = 0; i < N / 256; i++) {
        printf("%d+%d=%d\n", host_a[i], host_b[i], host_c[i]);
    }

```

```
*/
```

```
    cudaFreeHost(host_a);  
    cudaFreeHost(host_b);  
    cudaFreeHost(host_c);  
    cudaFree(dev_a0);  
    cudaFree(dev_b0);  
    cudaFree(dev_c0);  
    cudaFree(dev_a1);  
    cudaFree(dev_b1);  
    cudaFree(dev_c1);  
}
```

```
    cudaStreamDestroy(stream0);  
    cudaStreamDestroy(stream1);  
    return 0;  
}
```

Файл task3.h

```
#ifndef TASK3
```

```
#define TASK3
```

```
__global__ void mul_gpu(int* a, int* b, int* c);  
int task3();
```

```
#endif
```

Файл task3.cu

```
#include "task1.h"
```

```

#include "task2.h"

__global__ void mul_gpu(int* a, int* b, int* c) {

    int tid = threadIdx.x + blockIdx.x * blockDim.x;
    if (tid <= N / 256)
        c[tid] = abs(a[tid]) * abs(b[tid]) * a[tid] * b[tid] / (abs(a[tid]) * abs(b[tid]));
}

int task3() {
    //cuda-события для замера времени выполнения:
    cudaEvent_t start, stop;
    float elapsedTime;

    cudaStream_t stream0, stream1;
    cudaStreamCreate(&stream0);
    cudaStreamCreate(&stream1);

    printf("\n\tmultiplication of vectors through 2 streams:\n");
    for (long var_size = 1024; var_size <= FULL_DATA_SIZE / 2; var_size *= 2) {
        int* host_a, * host_b, * host_c;

        int* dev_a0, * dev_b0, * dev_c0; // первый gpu input buffer for stream0,
        который будет заполнен рандомными числами
        int* dev_a1, * dev_b1, * dev_c1; // второй gpu input buffer for stream1,
        который будет заполнен рандомными числами

        //выделение памяти на gpu
        cudaMalloc((void**)&dev_a0, var_size * sizeof(int));
        cudaMalloc((void**)&dev_b0, var_size * sizeof(int));
        cudaMalloc((void**)&dev_c0, var_size * sizeof(int));
    }
}

```

```

//выделение памяти на gpu
cudaMalloc((void**)&dev_a1, var_size * sizeof(int));
cudaMalloc((void**)&dev_b1, var_size * sizeof(int));
cudaMalloc((void**)&dev_c1, var_size * sizeof(int));

// выделение page-locked памяти, используемой для стримов
cudaHostAlloc((void**)&host_a, FULL_DATA_SIZE * sizeof(int),
cudaHostAllocDefault);

    cudaHostAlloc((void**)&host_b, FULL_DATA_SIZE * sizeof(int),
cudaHostAllocDefault);

    cudaHostAlloc((void**)&host_c, FULL_DATA_SIZE * sizeof(int),
cudaHostAllocDefault);


for (int i = 0; i < FULL_DATA_SIZE; i++) {
    host_a[i] = i;//rand();
    host_b[i] = i;// rand();
}


cudaEventCreate(&start);
cudaEventCreate(&stop);
cudaEventRecord(start, 0);


// теперь проитерировать всю дату, через байтные куски:
for (int i = 0; i < FULL_DATA_SIZE; i += var_size * 2) {
    // асинхронно копировать закрытую память на устройство:
    cudaMemcpyAsync(dev_a0, host_a + i, var_size * sizeof(int),
cudaMemcpyHostToDevice, stream0);

    cudaMemcpyAsync(dev_b0, host_b + i, var_size * sizeof(int),
cudaMemcpyHostToDevice, stream0);

    mul_gpu <<< var_size / 256, 256, 0, stream0 >>> (dev_a0, dev_b0, dev_c0);

```

```

        // копировать дату с устройства на закрытую памятьЖ
        cudaMemcpyAsync(host_c + i, dev_c0, var_size * sizeof(int),
cudaMemcpyDeviceToHost, stream0);

        // асинхронно копировать закрытую память на устройство
        cudaMemcpyAsync(dev_a1, host_a + i + var_size, var_size * sizeof(int),
cudaMemcpyHostToDevice, stream1);

        cudaMemcpyAsync(dev_b1, host_b + i + var_size, var_size * sizeof(int),
cudaMemcpyHostToDevice, stream1);

        mul_gpu <<< var_size / 256, 256, 0, stream1 >>> (dev_a1, dev_b1, dev_c1);

        // копировать дату с устройства на закрытую память
        cudaMemcpyAsync(host_c + i + var_size, dev_c1, var_size * sizeof(int),
cudaMemcpyDeviceToHost, stream1);
    }

    // синхронизируем оба потока
    cudaStreamSynchronize(stream0);
    cudaStreamSynchronize(stream1);
    cudaEventRecord(stop, 0);
    cudaEventSynchronize(stop);
    cudaEventElapsedTime(&elepsedTime, start, stop);

    printf("var_size = FULL_DATA_SIZE / %5ld. Time taken: %f ms\n",
FULL_DATA_SIZE / var_size, elepsedTime);

    //проверка:
    /*
    for (int i = 0; i < N / 256; i++) {
        printf("%d+%d=%d\n", host_a[i], host_b[i], host_c[i]);
    }
    */

```

```
}
```

```
*/
```

```
cudaFreeHost(host_a);
```

```
cudaFreeHost(host_b);
```

```
cudaFreeHost(host_c);
```

```
cudaFree(dev_a0);
```

```
cudaFree(dev_b0);
```

```
cudaFree(dev_c0);
```

```
cudaFree(dev_a1);
```

```
cudaFree(dev_b1);
```

```
cudaFree(dev_c1);
```

```
}
```

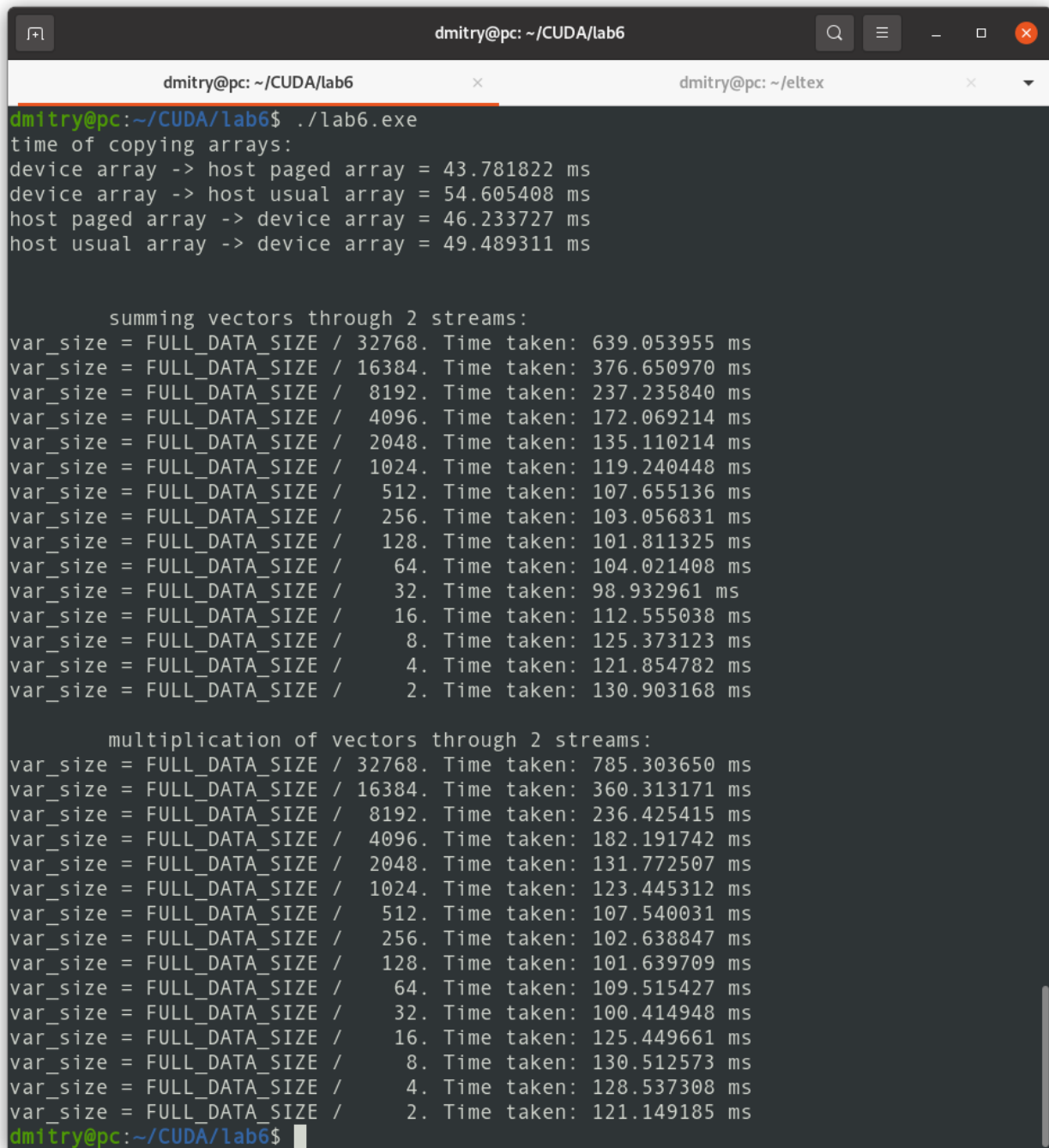
```
cudaStreamDestroy(stream0);
```

```
cudaStreamDestroy(stream1);
```

```
return 0;
```

```
}
```


Скриншоты



```
dmitry@pc: ~/CUDA/lab6$ ./lab6.exe
time of copying arrays:
device array -> host paged array = 43.781822 ms
device array -> host usual array = 54.605408 ms
host paged array -> device array = 46.233727 ms
host usual array -> device array = 49.489311 ms

summing vectors through 2 streams:
var_size = FULL_DATA_SIZE / 32768. Time taken: 639.053955 ms
var_size = FULL_DATA_SIZE / 16384. Time taken: 376.650970 ms
var_size = FULL_DATA_SIZE / 8192. Time taken: 237.235840 ms
var_size = FULL_DATA_SIZE / 4096. Time taken: 172.069214 ms
var_size = FULL_DATA_SIZE / 2048. Time taken: 135.110214 ms
var_size = FULL_DATA_SIZE / 1024. Time taken: 119.240448 ms
var_size = FULL_DATA_SIZE / 512. Time taken: 107.655136 ms
var_size = FULL_DATA_SIZE / 256. Time taken: 103.056831 ms
var_size = FULL_DATA_SIZE / 128. Time taken: 101.811325 ms
var_size = FULL_DATA_SIZE / 64. Time taken: 104.021408 ms
var_size = FULL_DATA_SIZE / 32. Time taken: 98.932961 ms
var_size = FULL_DATA_SIZE / 16. Time taken: 112.555038 ms
var_size = FULL_DATA_SIZE / 8. Time taken: 125.373123 ms
var_size = FULL_DATA_SIZE / 4. Time taken: 121.854782 ms
var_size = FULL_DATA_SIZE / 2. Time taken: 130.903168 ms

multiplication of vectors through 2 streams:
var_size = FULL_DATA_SIZE / 32768. Time taken: 785.303650 ms
var_size = FULL_DATA_SIZE / 16384. Time taken: 360.313171 ms
var_size = FULL_DATA_SIZE / 8192. Time taken: 236.425415 ms
var_size = FULL_DATA_SIZE / 4096. Time taken: 182.191742 ms
var_size = FULL_DATA_SIZE / 2048. Time taken: 131.772507 ms
var_size = FULL_DATA_SIZE / 1024. Time taken: 123.445312 ms
var_size = FULL_DATA_SIZE / 512. Time taken: 107.540031 ms
var_size = FULL_DATA_SIZE / 256. Time taken: 102.638847 ms
var_size = FULL_DATA_SIZE / 128. Time taken: 101.639709 ms
var_size = FULL_DATA_SIZE / 64. Time taken: 109.515427 ms
var_size = FULL_DATA_SIZE / 32. Time taken: 100.414948 ms
var_size = FULL_DATA_SIZE / 16. Time taken: 125.449661 ms
var_size = FULL_DATA_SIZE / 8. Time taken: 130.512573 ms
var_size = FULL_DATA_SIZE / 4. Time taken: 128.537308 ms
var_size = FULL_DATA_SIZE / 2. Time taken: 121.149185 ms
dmitry@pc: ~/CUDA/lab6$
```

Рисунок 1 — запуск программы

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

Память, выделенная с использованием закрепленных страниц, быстрее чем память, выделенная обычным образом.

В моем случае оптимальный размер порции данных для реализации сложения векторов с использованием потоков CUDA — это размер $1/32$ от всего размера вектора. Для умножения также $1/32$.