

**Филиал федерального государственного бюджетного  
образовательного учреждения высшего образования  
«Национальный исследовательский университет «МЭИ»  
в г. Смоленске**

Кафедра вычислительной техники

Направление: 09.04.01. «Информатика и вычислительная техника»  
Профиль: «Программное обеспечение средств вычислительной техники и  
автоматизированных систем»

Практическая работа №4  
«Отладка на кластере простых программ с использованием CUDA»  
по курсу:  
«Вычислительные системы»

Студент: Старостенков А.А.

Группа: ВМ-22(маг)

Вариант: 19

Преподаватель: Федулов А.С.

Смоленск, 2023

## Задание

1. Отладить, скомпилировать и запустить на гибридном вычислительном кластере (ГВК) СФМЭИ простые тестовые программы на языке Си с использованием технологии CUDA. В качестве тестовых программ использовать программы 1- 5 из лекции 5 по курсу «Вычислительные системы».
2. В программе 3 результат должен быть равен номеру по журналу.
3. В программах 4 и 5 каждый элемент результирующего вектора должен быть равен номеру по журналу.
4. В программе 5 построить зависимость времени вычисления суммы векторов от размера векторов. Печать результатов при больших значениях длины векторов ограничить 20-30 элементами результата.
5. Результат выполнения каждой программы дополнить кратким описанием (анализом, объяснением).
6. Предусмотреть вывод ФИО в каждой программе.
7. Выполнение программ показать преподавателю.
8. Все выполненные задания оформить в виде отчета.
9. Отчет должен содержать: тексты программ, команды компиляции и запуска, результаты запуска, объяснение результатов.
10. **Отчет в электронном виде** сдать преподавателю или направить на электронную почту: [director@sbmpei.ru](mailto:director@sbmpei.ru)

## 1. Программа 1

### Код

```
// Получение информации о некоторых свойствах устройств cuda
#include <cuda.h>
#include <stdio.h>
int main(int argc, char *argv[])
{
    printf("Starostenkov A.A. VM-22 (mag.)\n");
    printf("Example 1.\n");

    int deviceCount;
    cudaDeviceProp devProp;
    cudaGetDeviceCount(&deviceCount);
    printf("Found %d devices\n", deviceCount);
    for (int device = 0; device < deviceCount; device++)
    {
        cudaGetDeviceProperties(&devProp, device);
        printf("Device %d\n", device);
        printf("Compute capability      : %d.%d\n", devProp.major, devProp.minor);
        printf("Name                          : %s\n", devProp.name);
        printf("Total Global Memory            : %d\n", devProp.totalGlobalMem);
        printf("Shared memory per block: %d\n", devProp.sharedMemPerBlock);
        printf("Registers per block           : %d\n", devProp.regsPerBlock);
        printf("Warp size                      : %d\n", devProp.warpSize);
        printf("Max threads per block : %d\n", devProp.maxThreadsPerBlock);
        printf("Max threads dimensions: x = %d, y = %d, z = %d\n",
            devProp.maxThreadsDim[0],
            devProp.maxThreadsDim[1],
            devProp.maxThreadsDim[2]);
        printf("Max grid size: x = %d, y = %d, z = %d\n",
            devProp.maxGridSize[0],
            devProp.maxGridSize[1],
            devProp.maxGridSize[2]);
        printf("Clock rate: %d\n", devProp.clockRate);
        printf("Multiprocessor count: %d\n", devProp.multiProcessorCount);
        printf("Total constant memory : %d\n", devProp.totalConstMem);
    }

    return 0;
}
```

```
[starostenkov_aa@mng1 4]$ nvcc 1.cu -o 1
[starostenkov_aa@mng1 4]$ ./1
Starostenkov A.A. VM-22 (mag.)
Example 1.
Found 2 devices
Device 0
Compute capability      : 3.5
Name                    : GeForce GTX TITAN Black
Total Global Memory    : 2083782656
Shared memory per block: 49152
Registers per block     : 65536
Warp size               : 32
Max threads per block   : 1024
Max threads dimensions: x = 1024, y = 1024, z = 64
Max grid size: x = 2147483647, y = 65535, z = 65535
Clock rate: 980000
Multiprocessor count: 15
Total constant memory   : 65536
Device 1
Compute capability      : 3.5
Name                    : GeForce GTX TITAN Black
Total Global Memory    : 2084175872
Shared memory per block: 49152
Registers per block     : 65536
Warp size               : 32
Max threads per block   : 1024
Max threads dimensions: x = 1024, y = 1024, z = 64
Max grid size: x = 2147483647, y = 65535, z = 65535
Clock rate: 980000
Multiprocessor count: 15
Total constant memory   : 65536
[starostenkov_aa@mng1 4]$
```

Рисунок 1 – результат работы

## 2. Программа 2.

Код

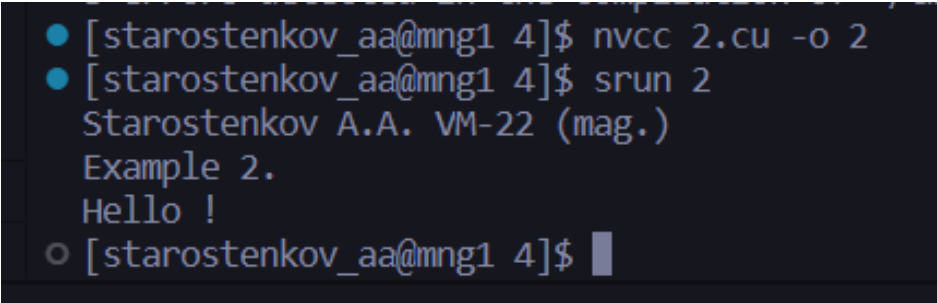


```
#include <cuda.h>
#include <stdio.h>

__global__ void kern(void)
{ // ничего не делает
}

int main()
{
    printf("Starostenkov A.A. VM-22 (mag.)\n");
    printf("Example 2.\n");

    kern<<1, 1>>>();
    printf("Hello !\n");
    return 0;
}
```



```
● [starostenkov_aa@mng1 4]$ nvcc 2.cu -o 2
● [starostenkov_aa@mng1 4]$ srun 2
Starostenkov A.A. VM-22 (mag.)
Example 2.
Hello !
○ [starostenkov_aa@mng1 4]$
```

Рисунок 2 – результат работы

3. В программе 3 результат должен быть равен номеру по журналу.

Код

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

__global__ void add(int *a, int *b, int *c) // Функция -
{
    *c = *a + *b;
}

int main(void)
{
    printf("Starostenkov A.A. VM-22 (mag.)\n");
    printf("Example 3.\n");

    int a, b, c; // host копии a, b, c
    int *dev_a, *dev_b, *dev_c; // device копии a, b, c
    int size = sizeof(int);
    // выделяем память для device копий для a, b, c
    cudaMalloc((void **)&dev_a, size);
    cudaMalloc((void **)&dev_b, size);
    cudaMalloc((void **)&dev_c, size);
    a = 10;
    b = 9;
    // копируем ввод на device
    cudaMemcpy(dev_a, &a, size, cudaMemcpyHostToDevice);
    cudaMemcpy(dev_b, &b, size, cudaMemcpyHostToDevice);
    // запускаем add() kernel на GPU, передавая параметры
    add<<<1, 1>>>>(dev_a, dev_b, dev_c);
    // копируем результат с на CPU
    cudaMemcpy(&c, dev_c, size, cudaMemcpyDeviceToHost);
    // освобождаем память device
    cudaFree(dev_a);
    cudaFree(dev_b);
    cudaFree(dev_c);
    printf("c = %5d\n", c);
    return 0;
}
```

```
● [starostenkov_aa@mng1 4]$ nvcc 3.cu -o 3
● [starostenkov_aa@mng1 4]$ srun 3
Starostenkov A.A. VM-22 (mag.)
Example 3.
c =      19
○ [starostenkov_aa@mng1 4]$
```

Рисунок 3 – результат работы

На видеокарте суммируются числа 10 и 9, получается 5.

4. В программах 4 и 5 каждый элемент результирующего вектора должен быть равен номеру по журналу.

Код

```
// Поэлементное сложение векторов (один блок)
#include <cuda.h>
#include <stdio.h>
#define N 128

__global__ void add(int *a, int *b, int *c)
{
    int tid = threadIdx.x; // Связываем элемент массива с глобальным номером нити
    if (tid > N - 1)
        return;
    c[tid] = a[tid] + b[tid]; // каждый tid – одна нить
}

int main()
{
    printf("Starostenkov A.A. VM-22 (mag.)\n");
    printf("Example 4.\n");

    int host_a[N], host_b[N], host_c[N];
    int *dev_a, *dev_b, *dev_c;
    for (int i = 0; i < N; i++)
    {
        host_a[i] = 10;
        host_b[i] = 9;
    }
    cudaMalloc((void **)&dev_a, N * sizeof(int));
    cudaMalloc((void **)&dev_b, N * sizeof(int));
    cudaMalloc((void **)&dev_c, N * sizeof(int));
    cudaMemcpy(dev_a, host_a, N * sizeof(int), cudaMemcpyHostToDevice);
    cudaMemcpy(dev_b, host_b, N * sizeof(int), cudaMemcpyHostToDevice);
    add<<<1, N>>>>(dev_a, dev_b, dev_c); // один блок, N потоков
    cudaMemcpy(host_c, dev_c, N * sizeof(int), cudaMemcpyDeviceToHost);
    for (int i = 0; i < N; i++)
    {
        printf("%d + %d = %d\n", host_a[i], host_b[i], host_c[i]);
    }
    cudaFree(dev_a);
    cudaFree(dev_b);
    cudaFree(dev_c);
    return 0;
}
```

```
10 + 9 = 19
10 + 9 = 19
10 + 9 = 19
10 + 9 = 19
10 + 9 = 19
10 + 9 = 19
10 + 9 = 19
10 + 9 = 19
10 + 9 = 19
○ [starostenkov_aa@mng1 4]$
```

Рисунок 4 – сумма векторов

На видеокарте суммируются два вектора по 128 элементов. Все элементы первого вектора содержат число 10, элементы второго – число 9. В результате получается вектор, каждый элемент которого содержит число 19.

5. В программе 5 построить зависимость времени вычисления суммы векторов от размера векторов. Печать результатов при больших значениях длины векторов ограничить 20-30 элементами результата.

Код

```
// Поэлементное сложение векторов (несколько блоков)
#include <cuda.h>
#include <stdio.h>

__global__ void add(int *a, int *b, int *c)
{
    // вычисление индекса массивов, используем несколько
    // блоков по координате x (blockDim.x),
    // в каждом блоке – несколько нитей по координате x
    int index = threadIdx.x + blockIdx.x * blockDim.x;
    c[index] = a[index] + b[index];
}

#define MIN_N (4096 * 4096 / 256) // Минимальный размер векторов
#define MAX_N (4096 * 4096 * 16) // Максимальный размер векторов
#define MULTIPLY_N 2 // множитель
#define THREADS_PER_BLOCK 1024 // максимальное значение для GK110

int main(void)
{
    printf("Starostenkov A.A. VM-22 (mag.)\n");
    printf("Example 5.\n");

    cudaEvent_t start, stop;
    float gpuTime;
    size_t n; // представления размера векторов

    printf("\n----START----\n");

    FILE *file = fopen("5_output.txt", "w");
    if (file == NULL)
    {
        printf("Failed to open the file.\n");
        return 1;
    }

    fprintf(file, "n: gpuTime\n");

    for (n = MIN_N; n <= MAX_N; n *= MULTIPLY_N)
    {
        printf("n = %d\n", n);

        int *a, *b, *c; // host копии a, b, c
        int *dev_a, *dev_b, *dev_c; // device копии of a, b, c
        // int size = N * sizeof(int);
        size_t size = n * sizeof(int);

        cudaEventCreate(&start);
        cudaEventCreate(&stop);

        // выделяем память на device для of a, b, c
        cudaMalloc((void **)&dev_a, size);
        cudaMalloc((void **)&dev_b, size);
        cudaMalloc((void **)&dev_c, size);
```

```

// выделяем память на хосте
a = (int *)malloc(size);
b = (int *)malloc(size);
c = (int *)malloc(size);
// инициализация массивов
size_t i;
for (i = 0; i < n; ++i)
    a[i] = 10;
for (i = 0; i < n; ++i)
    b[i] = 9;

cudaEventRecord(start, 0); // отсечка
// копируем ввод на device
cudaMemcpy(dev_a, a, size, cudaMemcpyHostToDevice);
cudaMemcpy(dev_b, b, size, cudaMemcpyHostToDevice);

// запускаем на выполнение add() kernel с блоками и нитями
add<<<n / THREADS_PER_BLOCK, THREADS_PER_BLOCK>>>(dev_a, dev_b,
dev_c);
// копируем результат работы device на host ( копия с )
cudaMemcpy(c, dev_c, size, cudaMemcpyDeviceToHost);
cudaEventRecord(stop, 0); // отсечка
cudaEventSynchronize(stop);
cudaEventElapsedTime(&gpuTime, start, stop);

printf("c = [");
for (i = 0; i < 20; ++i) printf("%d; ", c[i]);
printf("]\n");

printf("time spent executing by the GPU: %.2f milliseconds\n", gpuTime);

fprintf(file, "%zu: %.2f\n", n, gpuTime);

cudaEventDestroy(start);
cudaEventDestroy(stop);

free(a);
free(b);
free(c);
cudaFree(dev_a);
cudaFree(dev_b);
cudaFree(dev_c);
}

fclose(file);

return 0;
}

```



```

[starostenkov_aa@mng1 4]$ nvcc 5.cu -o 5
[starostenkov_aa@mng1 4]$ srun 5
Starostenkov A.A. VM-22 (mag.)
Example 5.

-----START-----
n = 65536
c = [19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; ]
time spent executing by the GPU: 0.64 milliseconds
n = 131072
c = [19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; ]
time spent executing by the GPU: 1.00 milliseconds
n = 262144
c = [19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; ]
time spent executing by the GPU: 1.98 milliseconds
n = 524288
c = [19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; ]
time spent executing by the GPU: 3.26 milliseconds
n = 1048576
c = [19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; ]
time spent executing by the GPU: 5.06 milliseconds
n = 2097152
c = [19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; ]
time spent executing by the GPU: 9.27 milliseconds
n = 4194304
c = [19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; ]
time spent executing by the GPU: 19.20 milliseconds
n = 8388608
c = [19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; ]
time spent executing by the GPU: 37.33 milliseconds
n = 16777216
c = [19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; ]
time spent executing by the GPU: 73.41 milliseconds
n = 33554432
c = [19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; ]
time spent executing by the GPU: 145.39 milliseconds
n = 67108864
c = [19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; ]
time spent executing by the GPU: 289.15 milliseconds
n = 134217728
c = [19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; ]
time spent executing by the GPU: 524.39 milliseconds
n = 268435456
c = [19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; ]
time spent executing by the GPU: 1047.32 milliseconds
[starostenkov_aa@mng1 4]$

```

Рисунок 5 – вычисление векторов

Ниже представлен файл «5\_output.txt» с записанными результатами для построения зависимости времени вычисления суммы векторов от размера векторов.

```

> 4 > 5_output.txt
n: gpuTime
65536: 0.64
131072: 1.00
262144: 1.98
524288: 3.26
1048576: 5.06
2097152: 9.27
4194304: 19.20
8388608: 37.33
16777216: 73.41
33554432: 145.39
67108864: 289.15
134217728: 524.39
268435456: 1047.32

```

Рисунок 6 – Зависимость времени вычисления суммы векторов от размера векторов

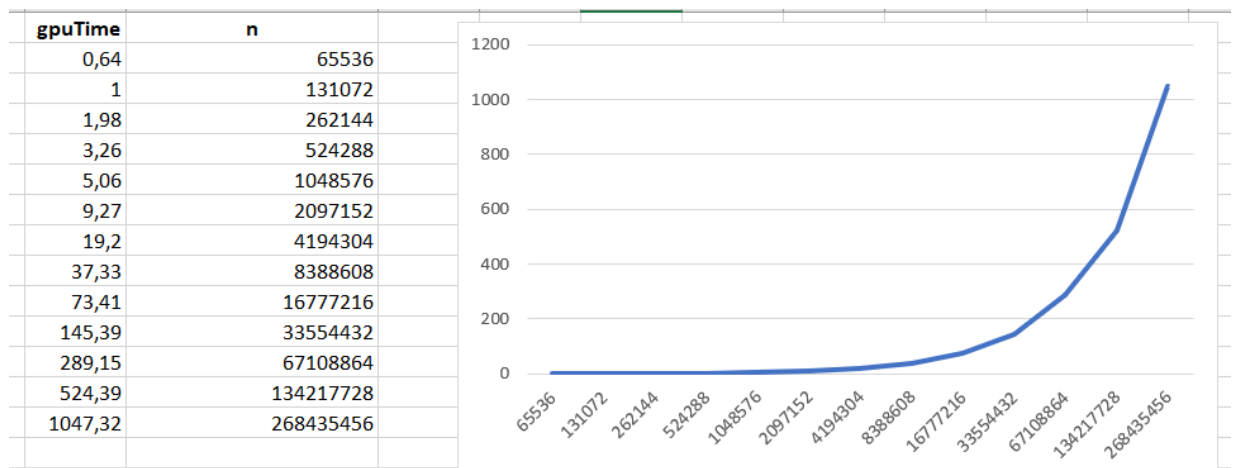


График 1 - Зависимость

Программа работает корректно. Время вычисления суммы векторов сильно зависит от размера вектора.