Отчет по лабораторной работе №1

Цель работы

Изучить модель программирования в CUDA, иерархию памяти, а также разработать программы для выполнения вычислений на GPU.

Задание 1

В MS Visual Studio создать проект CUDA VS Wizard. Ознакомиться и запустить программу «Hello world». Получить информацию об устройстве. Измерить время выполнения программы. Результаты занести в отчёт. Запустить программу «Hello world» на всех мультипроцессорах в GPU. Измерить время выполнения программы. Результаты занести в отчёт.

Код

```
#include "cuda_runtime.h"
#include "device_launch_parameters.h"

#include <stdio.h>
#include <math.h>

#define N (1024*1024)

class CudaTimer
{
public:
    CudaTimer()
    {
        cudaEventCreate(&start);
        cudaEventCreate(&stop);
        cudaEventRecord(start);
    }

    ~CudaTimer()
    {
      cudaEventRecord(stop);
      cudaEventSynchronize(stop);

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

      printf("Время выполнения: %.5f мс\n", elapsedTime);

      cudaEventDestroy(start);
      cudaEventDestroy(stop);
    }

private:
```

```
    cudaEvent_t start;
    cudaEvent_t stop;
};

__global__ void kernel(float * data)
{
   int idx     = blockIdx.x * blockDim.x + threadIdx.x;
   float x     = 2.0f * 3.1415926f * (float) idx / (float) N;
   data[idx]   = sinf(sqrtf(x));
}

void getInfo()
{
   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    : %u\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 ( "Total constant memory  : %d\n", devProp.totalConstMem );
   }

}

int main(int argc, char *argv[])
{
   getInfo();

   float* a    = (float*)malloc(N * sizeof(float));
   float* dev  = nullptr;

   cudaMalloc ((void**)&dev, N * sizeof(float));

   {
      CudaTimer t;
      kernel<<<dim3((N/512),1), dim3(512,1)>>> (dev);
   }

   cudaMemcpy(a, dev, N * sizeof(float), cudaMemcpyDeviceToHost);
```

```
    free(a);
    cudaFree(dev);

    return 0;
}
```

Результаты

```
Found 1 devices
Device 0
Compute capability     : 6.1
Name                   : NVIDIA GeForce GTX 1070
Total Global Memory    : 4203806720
Shared memory per block: 49152
Registers per block    : 65536
Warp size              : 32
Max threads per block  : 1024
Total constant memory  : 65536
Запуск на threads: 512, blocks: 512
Время выполнения: 0.38285 мс
```

```
Found 1 devices
Device 0
Compute capability     : 6.1
Name                   : NVIDIA GeForce GTX 1070
Total Global Memory    : 4203806720
Shared memory per block: 49152
Registers per block    : 65536
Warp size              : 32
Max threads per block  : 1024
Total constant memory  : 65536
Запуск на threads: 1024, blocks: 1024
Время выполнения: 0.29082 мс
```

Задание 2 (0 вариант)

Даны матрицы A и B из NxN натуральных (ненулевых) элементов (задаются случайно). Матрицы расположены в глобальной памяти. Написать программу, выполняющую перемножение двух матриц на GPU.

Код

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

```
#define BLOCK_SIZE 256

class CudaTimer
{
public:
    CudaTimer()
    {
        cudaEventCreate(&start);
        cudaEventCreate(&stop);
        cudaEventRecord(start);
    }

    ~CudaTimer()
    {
        cudaEventRecord(stop);
        cudaEventSynchronize(stop);

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

        printf("Время выполнения на GPU: %.5f мс\n", elapsedTime);
    }

private:
    cudaEvent_t start;
    cudaEvent_t stop;
};

__global__ void vectorMultiply(const int *A, const int *B, int *C, int N) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < N) {
        C[idx] = A[idx] * B[idx];
    }
}

void verifyResults(const int *A, const int *B, const int *C, int N) {
    for (int i = 0; i < N; i++)
    {
        if (C[i] != A[i] * B[i])
        {
            printf("Ошибка в элементе %d: %d * %d != %d\n", i, A[i], B[i],
C[i]);
            return;
        }
    }
    printf("Результаты верны!\n");
}

int main() {
    const int N = 1 << 20;
    size_t bytes = N * sizeof(int);

    int *h_A = (int *)malloc(bytes);
```

```
    int *h_B = (int *)malloc(bytes);
    int *h_C = (int *)malloc(bytes);

    srand(time(NULL));
    for (int i = 0; i < N; i++)
    {
        h_A[i] = rand() % 100 + 1;
        h_B[i] = rand() % 100 + 1;
    }

    int *d_A, *d_B, *d_C;
    cudaMalloc(&d_A, bytes);
    cudaMalloc(&d_B, bytes);
    cudaMalloc(&d_C, bytes);

    cudaMemcpy(d_A, h_A, bytes, cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, h_B, bytes, cudaMemcpyHostToDevice);

    dim3 threads(BLOCK_SIZE);
    dim3 blocks((N + BLOCK_SIZE - 1) / BLOCK_SIZE);

    {
        CudaTimer t;
        vectorMultiply<<<blocks, threads>>>(d_A, d_B, d_C, N);
    }

    cudaMemcpy(h_C, d_C, bytes, cudaMemcpyDeviceToHost);

    verifyResults(h_A, h_B, h_C, N);

    free(h_A);
    free(h_B);
    free(h_C);

    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_C);

    return 0;
}
```

Результаты

Размер векторов	Время выполнения
1048576 (2^20)	0.20435 мс
33554432 (2^25)	2.13907 мс

Ответы на контрольные вопросы

1. Что такое гибридное программирование?

Гибридное программирование — это подход к созданию программ для гетерогенных вычислительных систем, где задействуются как центральные процессоры (CPU), так и графические процессоры (GPU). CPU выполняет последовательные части программы, а GPU берет на себя массивно-параллельные вычисления, что позволяет добиться высокой производительности при решении задач.

2. Что такое CUDA?

CUDA (Compute Unified Device Architecture) — это программно-аппаратная платформа от NVIDIA для разработки приложений, использующих массивно-параллельные вычисления на графических процессорах. Она включает:

- Расширения языка C/C++ для написания кода, который запускается на GPU.
- API для управления памятью, потоками и взаимодействием между CPU и GPU.

3. Основные положения программной модели CUDA?

1. Иерархия потоков:
    - В CUDA задачи распределяются между нитями (threads), которые рганизованы в блоки (blocks).
    - Блоки, в свою очередь, формируют сетку (grid).
2. Память:
    - В CUDA существует несколько видов памяти: глобальная, разделяемая (shared), локальная, текстурная и константная.
    - Нити одного блока могут обмениваться данными через разделяемую память.
3. Код:
    - Код состоит из двух частей: последовательной (на CPU) и параллельной (на GPU).
    - Ядро CUDA (kernel) — это функция, которая выполняется параллельно всеми нитями.

4. Из чего состоит программный стек CUDA?

Программный стек CUDA включает:

1. CUDA Runtime API: Высокоуровневый интерфейс для управления памятью, потоками и выполнения ядра.
2. CUDA Driver API: Низкоуровневый интерфейс, обеспечивающий прямой контроль над GPU.
3. Библиотеки:

- cuBLAS — для линейной алгебры.
- cuFFT — для преобразования Фурье.
- cuDNN — для нейронных сетей и глубокого обучения.

4. Драйверы GPU: Обеспечивают взаимодействие между программами и оборудованием.
5. Инструменты разработки:
    - Компилятор nvcc.
    - Профилировщики (Nsight).

5. Что такое ядро в CUDA?

Ядро (kernel) — это функция, которая выполняется параллельно на GPU большим количеством потоков. Ядро запускается из кода на CPU с указанием конфигурации сетки и блоков. Каждый поток обрабатывает определенный элемент данных, используя уникальные идентификаторы (threadIdx, blockIdx).

6. Какие расширения языка C вводятся в CUDA?

Спецификаторы функций:

- **global** — для функций-ядер, которые вызываются с CPU и исполняются на GPU.
- **device** — для функций, которые исполняются только на GPU и вызываются из других функций GPU.
- **host** — для функций, исполняющихся на CPU (по умолчанию).

Спецификаторы памяти:

- **shared** — для разделяемой памяти на уровне блока.
- **device** — для глобальной памяти GPU.
- **constant** — для константной памяти GPU.

Встроенные переменные: threadIdx, blockIdx, blockDim, gridDim. Типы данных: int3, float4, dim3.

7. Какие встроенные переменные поддерживаются в CUDA и для чего они нужны?

- threadIdx — индекс текущего потока в блоке.
- blockIdx — индекс текущего блока в сетке.
- blockDim — количество потоков в блоке.
- gridDim — количество блоков в сетке.
- warpSize — размер warp'а (обычно 32 потока). Эти переменные помогают потокам идентифицировать себя и работать с определенными данными.

8. Какие ограничения вводятся на функции, выполняемые на GPU?

1. Функции GPU не могут:
    - Использовать рекурсию.
    - Иметь переменное количество аргументов.
    - Возвращать значения (кроме **global** функций, возвращающих void).
2. Ограничения на переменные:
    - Нельзя использовать статические переменные внутри функций.
    - Переменные в **shared** инициализируются только с CPU.
3. Адрес функции взять нельзя (кроме **global**).
4. Указатели и сложные структуры данных работают с ограничениями.