МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт комплексной безопасности и цифровых технологий

Кафедра КБ-14 «Цифровые технологии обработки данных»

Платформы анализа больших данных

Лабораторная работа 6

Вариант 8

Выполнил:

Студент группы БСБО-09-22

Шутов Кирилл Сергеевич

Проверил:

Кашкин Евгений Владимирович

Москва, 2025

Постановка задачи

Реализовать операцию поэлементного сложения двух векторов на GPU в среде CUDA и OpenCL, сравнить подходы.

Описание кода и выполненных действий

Хост-код на OpenCL: выбор платформы/устройства, создание контекста/очереди, чтение и сборка программы из add.cl, создание буферов, передача аргументов, запуск, чтение результата и освобождение ресурсов.

```
#include <CL/cl.h>
#include <stdio.h>
#include <stdlib.h>
#define ARRAY_SIZE 5
int main()
{
   const int arraySize = ARRAY_SIZE;
   int a[ARRAY_SIZE] = { 1, 2, 3, 4, 5 };
   int b[ARRAY_SIZE] = { 10, 20, 30, 40, 50 };
   int c[ARRAY_SIZE] = { 0 };
   cl_int err;
   cl_platform_id platform;
   cl_device_id device;
   cl_context context;
   cl_command_queue queue;
   cl_program program;
   cl_kernel kernel;
   cl_mem bufA, bufB, bufC;
   err = clGetPlatformIDs(1, &platform, NULL);
   if (err != CL_SUCCESS) { fprintf(stderr, "clGetPlatformIDs -> %d\n", err);
return 1; }
   err = clGetDeviceIDs(platform, CL_DEVICE_TYPE_GPU, 1, &device, NULL);
   if (err != CL_SUCCESS) { fprintf(stderr, "clGetDeviceIDs -> %d\n", err); return
1; }
   context = clCreateContext(NULL, 1, &device, NULL, NULL, &err);
   if (err != CL_SUCCESS) { fprintf(stderr, "clCreateContext -> %d\n", err);
return 1; }
   cl_command_queue_properties properties[] = { 0 }; // Пустой массив свойств
   queue = clCreateCommandQueueWithProperties(context, device, properties, &err);
   if (err != CL_SUCCESS) {
       fprintf(stderr, "clCreateCommandQueueWithProperties -> %d\n", err);
       return 1;
   }
   if (err != CL_SUCCESS) { fprintf(stderr, "clCreateCommandQueue -> %d\n", err);
return 1; }
   const char* srcFilename = "add.cl";
   FILE* f = fopen(srcFilename, "r");
   if (!f) { perror("fopen"); return 1; }
   fseek(f, 0, SEEK_END);
   size_t sourceSize = ftell(f);
   rewind(f);
   char* sourceStr = malloc(sourceSize + 1);
   if (sourceStr == NULL) {
       fprintf(stderr, "Memory allocation failed\n");
       fclose(f);
       return 1;
   }
   fread(sourceStr, 1, sourceSize, f);
   sourceStr[sourceSize] = '\0';
   fclose(f);
```

```
    program = clCreateProgramWithSource(context, 1, (const char**)&sourceStr,
&sourceSize, &err);
    if (err != CL_SUCCESS) { fprintf(stderr, "clCreateProgramWithSource -> %d\n",
err); return 1; }
    free(sourceStr);
    err = clBuildProgram(program, 1, &device, NULL, NULL, NULL);
    if (err != CL_SUCCESS) {
        size_t logSize;
        clGetProgramBuildInfo(program, device, CL_PROGRAM_BUILD_LOG, 0, NULL,
&logSize);
        char* log = malloc(logSize);
        clGetProgramBuildInfo(program, device, CL_PROGRAM_BUILD_LOG, logSize, log,
NULL);
        fprintf(stderr, "Build error:\n%s\n", log);
        free(log);
        return 1;
    }
    kernel = clCreateKernel(program, "addKernel", &err);
    if (err != CL_SUCCESS) { fprintf(stderr, "clCreateKernel -> %d\n", err); return
1; }
    bufA = clCreateBuffer(context, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
        ARRAY_SIZE * sizeof(int), a, &err);
    bufB = clCreateBuffer(context, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
        ARRAY_SIZE * sizeof(int), b, &err);
    bufC = clCreateBuffer(context, CL_MEM_WRITE_ONLY,
        ARRAY_SIZE * sizeof(int), NULL, &err);
    clSetKernelArg(kernel, 0, sizeof(cl_mem), &bufA);
    clSetKernelArg(kernel, 1, sizeof(cl_mem), &bufB);
    clSetKernelArg(kernel, 2, sizeof(cl_mem), &bufC);
    size_t globalWorkSize = ARRAY_SIZE;
    err = clEnqueueNDRangeKernel(queue, kernel, 1, NULL,
        &globalWorkSize, NULL, 0, NULL, NULL);
    if (err != CL_SUCCESS) { fprintf(stderr, "clEnqueueNDRangeKernel -> %d\n",
err); return 1; }
    clFinish(queue);
    err = clEnqueueReadBuffer(queue, bufC, CL_TRUE, 0,
        ARRAY_SIZE * sizeof(int), c, 0, NULL, NULL);
    if (err != CL_SUCCESS) { fprintf(stderr, "clEnqueueReadBuffer -> %d\n", err);
return 1; }
    printf("{1,2,3,4,5} + {10,20,30,40,50} = {%d,%d,%d,%d,%d}\n",
        c[0], c[1], c[2], c[3], c[4]);
    clReleaseMemObject(bufA);
    clReleaseMemObject(bufB);
    clReleaseMemObject(bufC);
    clReleaseKernel(kernel);
    clReleaseProgram(program);
    clReleaseCommandQueue(queue);
    clReleaseContext(context);
    return 0;
}
```

Листинг 1. main.c

Сам кернел.

```
#include "cuda_runtime.h"
#include "device_launch_parameters.h"
#include <stdio.h>

__global__ void addKernel(const int* a, const int* b, int* c)
{
    int i = threadIdx.x + blockIdx.x * blockDim.x;
    c[i] = a[i] + b[i];
}
```

Листинг 2. add.cl

Таблица 1. Замеры

Фреймворк	Время GPU (мс)
CUDA	0.06
OpenCL	0.21

Сравнение способов, CUDA удобна и заточена под NVIDIA: быстро, просто, удобно, OpenCL универсальна, работает на любом железе, но требует много настройки. На простых задачах может быть даже чуть медленнее.

Вывод

Для проектов, где важна производительность на NVIDIA — **CUDA** лучший выбор. Если проект должен работать на разных платформах — тогда **OpenCL**.

Источники

1. Документация NVIDIA CUDA. [Электронный ресурс] URL: https://docs.nvidia.com/cuda/ Дата обращения: (03.03.2025 г).
2. Shared Memory Optimizations. [Электронный ресурс] URL: https://docs.nvidia.com/cuda/ Дата обращения: (03.04.2025 г).