

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Кафедра вычислительной математики и программирования

спецкурс «Параллельное программирование»

КУРСОВАЯ РАБОТА

Выполнил:	Алмазов М.С.
Группа:	18-2012
Преподаватель:	Семенов С.А.

Оглавление

Введение.....	4
Термины.....	5
Особенности языка.....	5
Освоение OpenCL.....	6
Постановка задачи.....	6
Описание решения.....	6
Основные моменты кода.....	6
Результат работы программы.....	7
Сравнение скорости выполнения на CPU и GPU.....	8
Выводы.....	8
AMD vs NVidia.....	9
Постановка задачи.....	9
Описание решения.....	9
Основные моменты кода.....	9
Результат работы программ.....	10
Сравнение эффективности видеокарт AMD Radeon R7 M340 и Nvidia GeForce GTX 950M.....	11
Выводы.....	11
Геометрические операции.....	13
Постановка задачи.....	13
Описание решения.....	13
Основные моменты кода.....	13
Результат выполнения программы.....	14
Выводы.....	15
Приложения.....	16
Код программы «Вычислить значение функции ».....	16
Код программы «Логарифм».....	18
Код программы «Геометрия».....	24

Введение

OpenCL – это фреймворк для написания ПО, связанного с параллельными вычислениями на различных графических и центральных процессорах, а также ПЛИС. OpenCL включает в себя язык программирования, основанный на стандарте C99, и является полностью открытым стандартом. OpenCL разрабатывается и поддерживается группой Khronos Group, в которую входят многие крупные коммерческие организации, включая AMD, Apple, ARM, Intel, Nvidia, Sony.

Изначально разработкой OpenCL занималась компания Apple, которая внесла предложения по разработке спецификации в комитет Khronos. Через некоторое время AMD решила принять участие в разработке фреймворка, который пришел бы на замену Close to Metal. В 2008 году была образована рабочая группа Khronos Compute для разработки спецификаций OpenCL, в которую вошли Apple, Nvidia, AMD, IBM, Intel, ARM, Motorola и другие организации, в т.ч. занимающиеся разработкой развлекательного ПО. В том же году выпущена первая версия стандарта, а в апреле 2009 – бета-версии SDK от AMD и Nvidia.

Основной задачей курсовой работы является овладение навыками разработки приложений для процессоров с массивно параллельной вычислительной архитектурой, способных выполнять более чем 64 арифметические операции за один цикл тактовой частоты. Специализированное программирование таких процессоров требует детального понимания принципов параллельного программирования, а также моделей параллелизма, типов памяти GPU, условий обмена данными с CPU и знания имеющихся архитектурных ограничений этих процессоров.

В курсовой работе изучены различные алгоритмы решения задач и проведен сравнительный анализ быстродействия этих алгоритмов, выполненных на центральном процессоре и видеокарте. Дополнительно проведено сравнение производительности GPU AMD и NVidia одной ценовой категории. В качестве целевого исследования ставилось определение того, какое средство можно считать наиболее эффективным по производительности в решении тех или иных задач: интегральная схема центрального процессора (CPU) или

возможности параллельных архитектур процессоров устройств графических карт (GPU).

Термины

- Host — устройство, управляющее выполнением программы
- Device — устройство, на котором выполняется программа
- Compute Unit — блок, состоящий из 16 параллельных вычислительных устройств (Stream Core)
- Stream Core — блок, состоящий из 5 вычислительных модулей (Processing Element) и регистров общего назначения.
- Kernel — код, исполняемый на устройстве (Device)
- Work item — одна многопоточная задача, обрабатываемая при помощи Kernel
- Work group — группа многопоточных задач (Work item)

Особенности языка

- Отсутствие поддержки указателей на функции, рекурсии, битовых полей, массивов переменной длины, стандартных заголовочных файлов.
- Иной набор встроенных функций
- Квалификаторы типов памяти: `__global`, `__local`, `__constant`, `__private`
- Расширения языка для параллельных вычислений: векторные типы, синхронизация, функции для работы с work group и work item

Освоение OpenCL

Постановка задачи

Вычислить значение функции $\frac{\sin(x)}{x}$. В качестве значения аргумента функции решено использовать псевдослучайные числа.

Описание решения

Вычисление значения на GPU выполняется специальной функцией. Она берет значение аргумента из массива данных. Далее вычисляется значение математической функции, после чего оно помещается в массив данных по тому индексу, откуда был взят аргумент.

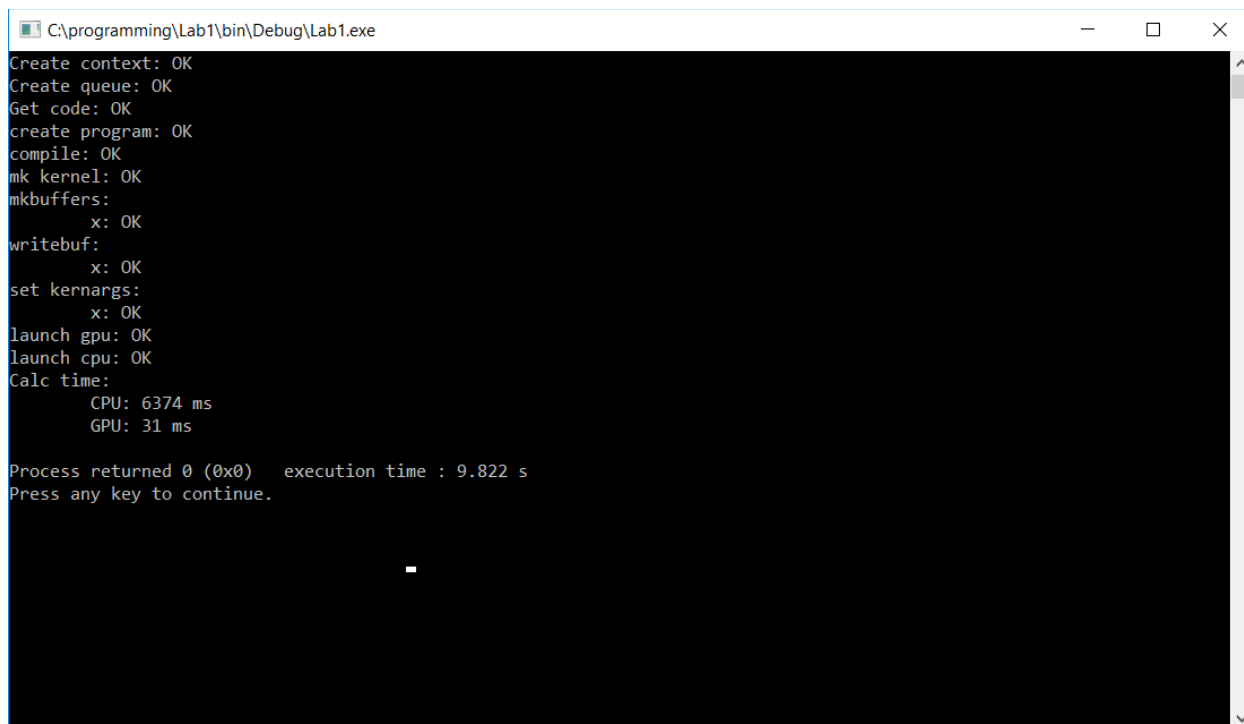
Основные моменты кода

Массив `_x[]` заполняется вычисляемыми значениями синуса поэлементно каждым отдельным ядром GPU в функции `test(float *_x)`:

```
float x = _x[id];  
float ret = sin(x)/x;  
_x[id] = ret;
```

где `id` - номер обрабатываемого значения.

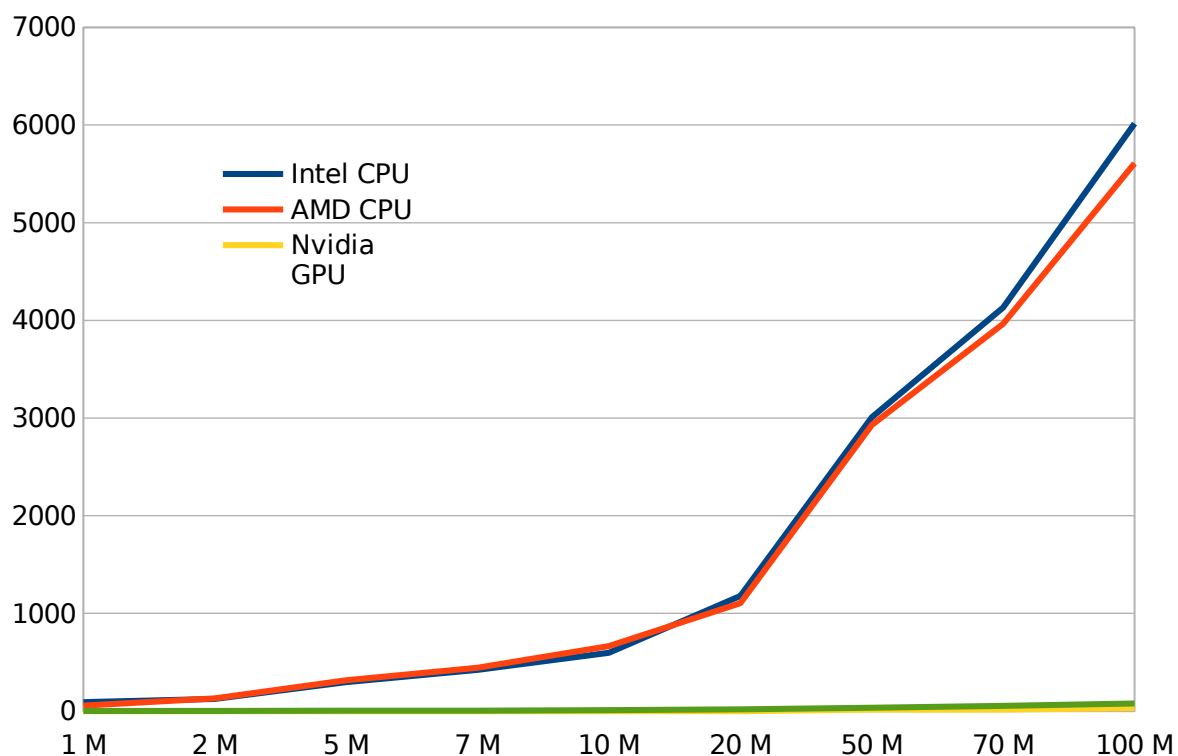
Результат работы программы



```
C:\programming\Lab1\bin\Debug\Lab1.exe
Create context: OK
Create queue: OK
Get code: OK
create program: OK
compile: OK
mk kernel: OK
mkbuffers:
    x: OK
writebuf:
    x: OK
set kernargs:
    x: OK
launch gpu: OK
launch cpu: OK
Calc time:
    CPU: 6374 ms
    GPU: 31 ms
Process returned 0 (0x0)  execution time : 9.822 s
Press any key to continue.
-
```

Рис. 1. Окно вывода консольного приложения

Сравнение скорости выполнения на CPU и GPU



Сравнение времени выполнения программы на CPU и GPU при разном количестве входных данных

Из графика видно, что GPU справляется с задачей намного быстрее, чем CPU, особенно при большом объеме входных данных.

Выводы

Несмотря на неизбежные накладные расходы, связанные с необходимостью разбиения задачи и пересылки данных, выигрыш в производительности довольно существенен при использовании в программе OpenCL в сравнении с традиционным алгоритмом решения на CPU.

AMD vs NVidia

Постановка задачи

Выполнить параллельные вычисления на видеокарте **Nvidia GeForce GTX 950M**, сравнить с эффективностью работы видеокарты **AMD Radeon R7 M340**, используя в обоих случаях один и тот же пример решения задачи вычисления значения функции $\frac{\sin(x)}{x}$ при заданном значении аргумента.

Описание решения

В качестве входного демонстрационного примера для сравнения эффективности вычислений видеокарт **Nvidia GeForce GTX 950M** и **AMD Radeon R7 M340** используется программа вычисления значения указанной функции с количеством входных, а соответственно и выходных данных принятым $N = 100$ млн.

Значение аргумента функции берется псевдослучайное.

Основные моменты кода

Массив `_x[]` заполняется вычисляемыми значениями функции поэлементно каждым отдельным ядром GPU в функции `test(float * _x)`:

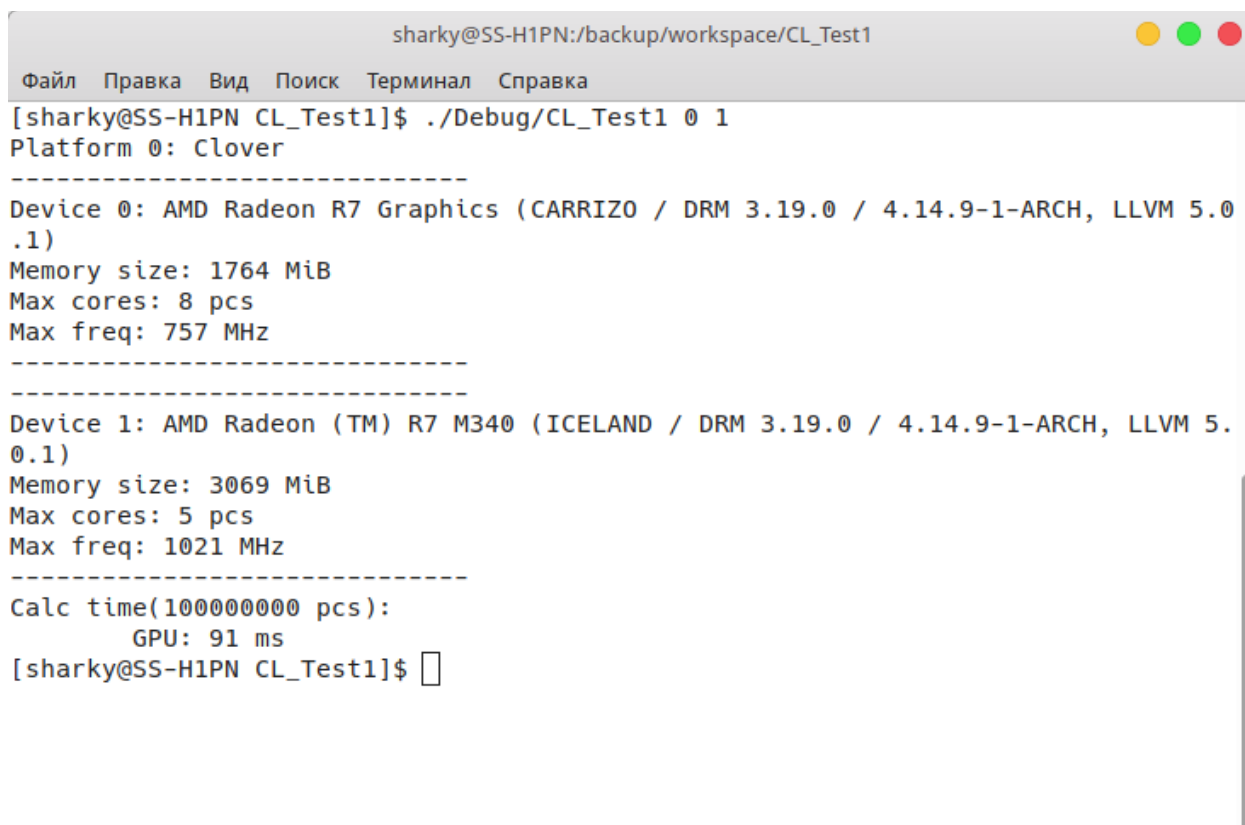
```
__kernel void test(__global float* _x)
{
    int id = get_global_id(0);
    float x = _x[id];
    float ret = sin(x)/x;
    _x[id] = ret;
}
```

В основном коде программы определяется время исполнения задачи на устройстве.

Дополнительно программа выводит параметры видеокарты, такие как:

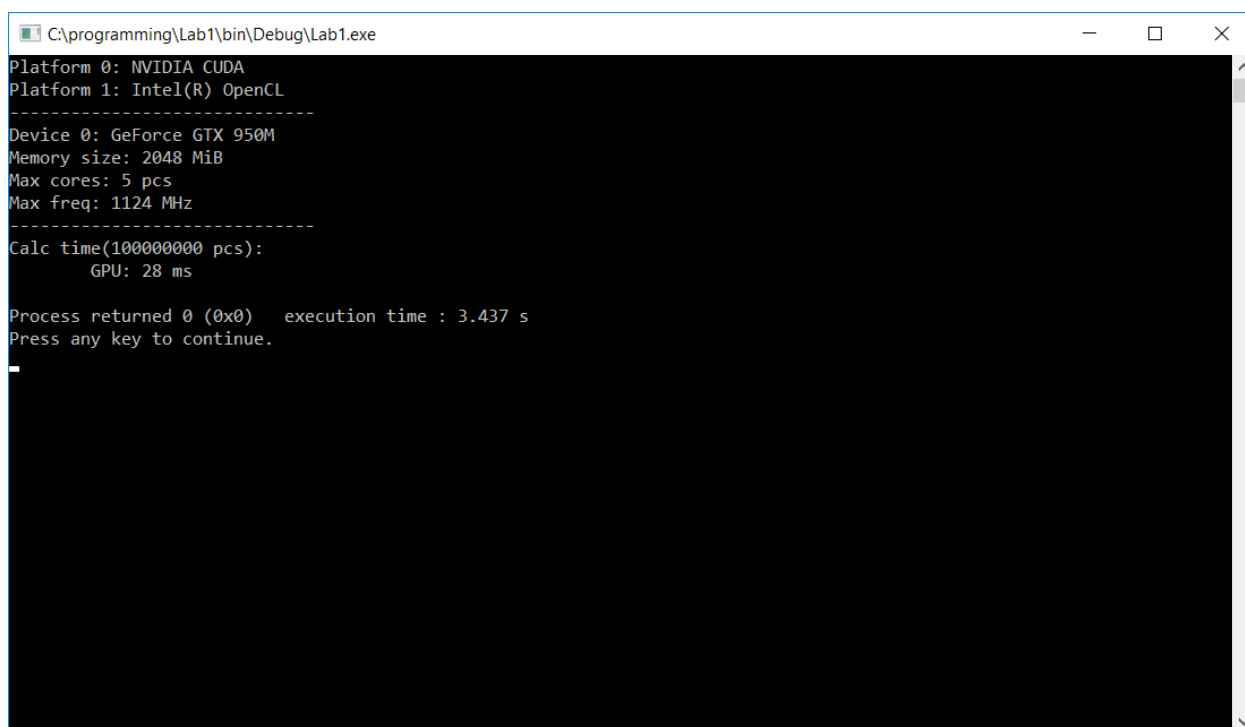
- имя устройства;
- объем памяти;
- максимальное количество вычислительных ядер;
- тактовая частота;

Результат работы программ



```
sharky@SS-H1PN:/backup/workspace/CL_Test1
Файл  Правка  Вид  Поиск  Терминал  Справка
[sharky@SS-H1PN CL_Test1]$ ./Debug/CL_Test1 0 1
Platform 0: Clover
-----
Device 0: AMD Radeon R7 Graphics (CARRIZO / DRM 3.19.0 / 4.14.9-1-ARCH, LLVM 5.0.1)
Memory size: 1764 MiB
Max cores: 8 pcs
Max freq: 757 MHz
-----
Device 1: AMD Radeon (TM) R7 M340 (ICELAND / DRM 3.19.0 / 4.14.9-1-ARCH, LLVM 5.0.1)
Memory size: 3069 MiB
Max cores: 5 pcs
Max freq: 1021 MHz
-----
Calc time(1000000000 pcs):
      GPU: 91 ms
[sharky@SS-H1PN CL_Test1]$
```

Рис. 11. Окно вывода консольного приложения, выполненного **AMD Radeon R7 M340**



```
C:\programming\Lab1\bin\Debug\Lab1.exe
Platform 0: NVIDIA CUDA
Platform 1: Intel(R) OpenCL
-----
Device 0: GeForce GTX 950M
Memory size: 2048 MiB
Max cores: 5 pcs
Max freq: 1124 MHz
-----
Calc time(100000000 pcs):
      GPU: 28 ms
Process returned 0 (0x0)   execution time : 3.437 s
Press any key to continue.
```

Рис. 12. Окно вывода консольного приложения, выполненного **Nvidia GeForce GTX 950M**

Сравнение эффективности видеокарт AMD Radeon R7 M340 и Nvidia GeForce GTX 950M

При запуске одной и той же программы на исполнение в двух вариантах использования ускорителей были получены следующие результаты:

	Radeon R7 M340	GeForce GTX 950M
Время выполнения	91 мс	28 мс
Тактовая частота	1021 МГц	1124 МГц
Объем оперативной памяти	3069 МиБ	2048 МиБ
Число ядер	5	5

Сравнение времени выполнения вычислений графических ускорителей при различных объемах заданий:

Количество элементов массива	Radeon R7 M340	GeForce GTX 950M
	t, мс	t, мс
25 млн	23	8
50 млн	45	15
75 млн	69	24
100 млн	94	31
250 млн	180	-

Выводы

В данной работе проведен анализ производительности видеокарт AMD Radeon R7 M340 и Nvidia GeForce GTX 950M на примере исполнения одной и той же программы вычислений.

При сравнении реализации задачи на CPU и GPU, несмотря на неизбежные накладные расходы, связанные с необходимостью разбиения задачи и пересылкой данных, выигрыш в производительности довольно существенен при использовании ускорителей в обоих случаях. Кроме того, следует особо подчеркнуть, что в результате исследований выявлена более высокая производительность GPU Nvidia, время выполнения вычислений на нем в среднем в 3 раза меньше, чем на GPU AMD.

Но по результатам исследований выявлено, что использование GPU AMD целесообразно на задачах, требующих объемов вычислений,

превышающих возможности NVidia. Так, например, при количестве значений выше 100 млн NVidia уже не способна выполнить вычисления из-за нехватки графической памяти, AMD же решает задачи и более сложные.

Нельзя не обратить внимание также и на тот факт, что в исследовании использовались GPU для портативных устройств, при проектировании которых большее внимание уделяется не высокой производительности, а низкому энергопотреблению. В интернете можно найти сравнительные тесты, показывающие, что наиболее высокопроизводительные GPU от AMD все же превосходят GPU Nvidia по вычислительным возможностям.

Геометрические операции

Постановка задачи

Выполнить параллельные вычисления на GPU, определяющие точку пересечения луча и треугольника.

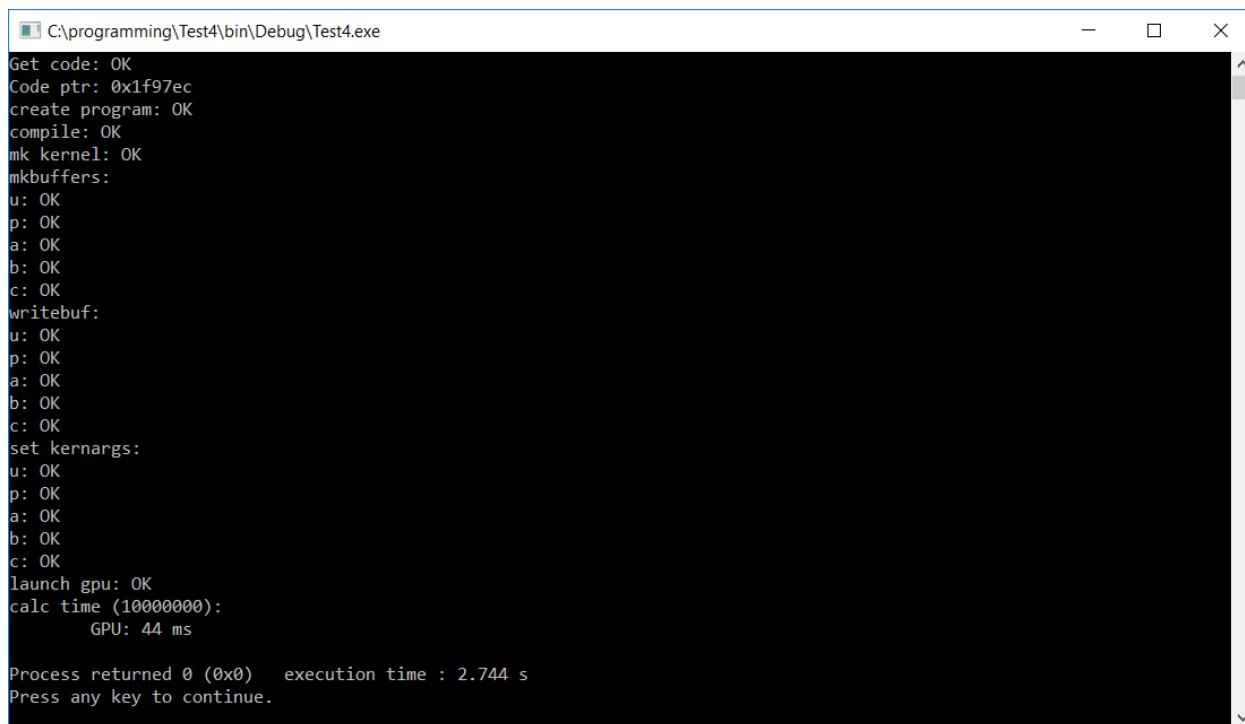
Описание решения

Вычисление значения на GPU выполняется специальной функцией. Она берет координаты точек треугольника, начала луча и вектор направления луча из специально отведенных массивов. Далее вычисляются координаты точки пересечения луча и треугольника, если это возможно. В случае успеха в дополнительной переменной фиксируется этот факт путем занесения специального числа. Если не удалось найти точку пересечения, то в эту дополнительную переменную записывается другое магическое значение, исходя из которого можно понять причину неудачи (луч параллелен плоскости треугольника, или луч не попадает в треугольник, или луч направлен от плоскости). Дополнительная переменная и координаты точки пересечения сохраняются в специальные массивы, в которых до выполнения операции хранятся входные данные.

Основные моменты кода

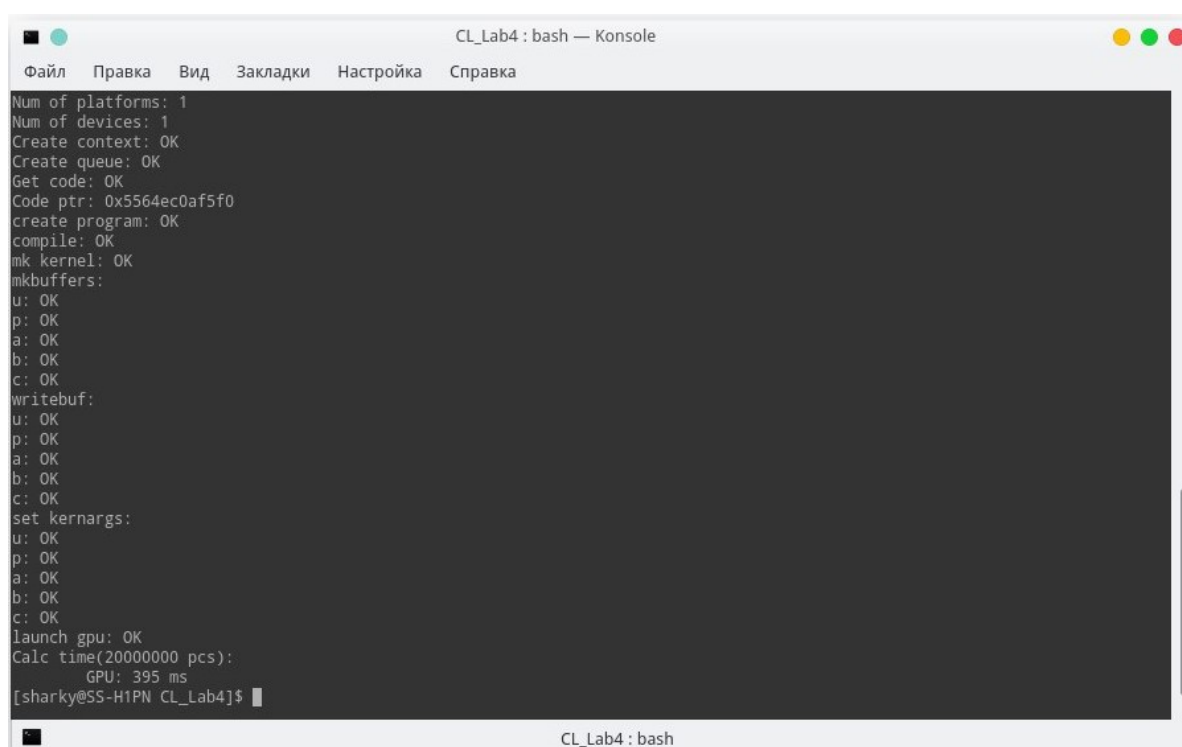
Массив точек пересечения заполняется вычисляемыми значениями поэлементно каждым отдельным ядром GPU в функции `test`. Для повышения читаемости и снижения громоздкости кода некоторые фрагменты, такие, как скалярное и векторное произведение векторов, например, вынесены в отдельные функции.

Результат выполнения программы



```
C:\programming\Test4\bin\Debug\Test4.exe
Get code: OK
Code ptr: 0x1f97ec
create program: OK
compile: OK
mk kernel: OK
mkbuffers:
u: OK
p: OK
a: OK
b: OK
c: OK
writebuf:
u: OK
p: OK
a: OK
b: OK
c: OK
set kernargs:
u: OK
p: OK
a: OK
b: OK
c: OK
launch gpu: OK
calc time (10000000):
GPU: 44 ms
Process returned 0 (0x0)  execution time : 2.744 s
Press any key to continue.
```

Окно вывода консольного приложения (Windows)



```
CL_Lab4 : bash — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
Num of platforms: 1
Num of devices: 1
Create context: OK
Create queue: OK
Get code: OK
Code ptr: 0x5564ec0af5f0
create program: OK
compile: OK
mk kernel: OK
mkbuffers:
u: OK
p: OK
a: OK
b: OK
c: OK
writebuf:
u: OK
p: OK
a: OK
b: OK
c: OK
set kernargs:
u: OK
p: OK
a: OK
b: OK
c: OK
launch gpu: OK
Calc time(20000000 pcs):
GPU: 395 ms
[sharky@SS-H1PN CL_Lab4]$
```

Окно вывода консольного приложения (Linux)

Выводы

При проведении измерений под ОС Windows(c)(r)TM наблюдается прирост в производительности, который можно объяснить применением проприетарной реализацией библиотеки OpenCL. Также GPU Nvidia показал меньшее время выполнения программы.

Приложения

Код программы «Вычислить значение функции $\frac{\sin(x)}{x}$ »

Код C++ :

```
#include <iostream>
#include <string>
#define CL_USE_DEPRECATED_OPENCL_1_2_APIS
#include <CL/cl.h>
#include <stdio.h>
#include <cmath>
#include <cstdlib>
#include <chrono>

using namespace std;

float get_val(float x)
{
    float ret = sin(x)/x;
    return ret;
}

void calculate(float * X, size_t N)
{
    size_t i = 0;
    while(i<N)
    {
        float x = X[i];
        X[i] = get_val(x);
        i++;
    }
}

int main()
{
    /* получить доступные платформы */
    cl_platform_id platform_id;
    unsigned int ret_num_platforms;
    cl_int ret = clGetPlatformIDs(1, &platform_id, &ret_num_platforms);
    cout << "Num of platforms: " << ret_num_platforms << "\n";

    /* получить доступные устройства */
    cl_device_id device_id;
    unsigned int ret_num_devices;
    ret = clGetDeviceIDs(platform_id, CL_DEVICE_TYPE_DEFAULT, 1, &device_id,
    &ret_num_devices);
    cout << "Num of devices: " << ret_num_devices << "\n";
    /* создать контекст */
    cl_context context = clCreateContext(NULL, 1, &device_id, NULL, NULL, &ret);
    cout << "Create context: " << clke_descr(ret) << "\n";
    /* создаем команду */
    cl_command_queue command_queue = clCreateCommandQueue(context, device_id, 0,
    &ret);
    cout << "Create queue: " << clke_descr(ret) << "\n";

    cl_program program = NULL;
    cl_kernel kernel = NULL;
    FILE *fp;
    const char fileName[] = "src/test.ocl";
    size_t source_size;
```



```

string source_str = "";
try
{
    fp = fopen(fileName, "r");
    if (!fp)
    {
        fprintf(stderr, "Failed to load kernel.\n");
        return 1;
    }
    //source_str = (char *)malloc(MAX_SOURCE_SIZE);
    while (!feof(fp))
    {
        char c;
        fread(&c, 1, 1, fp);
        source_str += c;
    }
    fclose(fp);
}
catch (...)
{
    printf("EXCEPTION\n");
}
cout << "Get code: ";
const char *sstrptr = source_str.c_str();
source_size = source_str.length();

/* создать бинарник из кода программы */
cout << "OK\n";
//cout << "Code ptr: " << (void *)sstrptr << "\n";
program = clCreateProgramWithSource(context, 1, &sstrptr, (const size_t *)
&source_size, &ret);
cout << "create program: " << clke_descr(ret) << "\n";
/* скомпилировать программу */

ret = clBuildProgram(program, 1, &device_id, NULL, NULL, NULL);
cout << "compile: " << clke_descr(ret) << "\n";
if (ret) return -110;
/* создать кернел */
kernel = clCreateKernel(program, "test", &ret);
cout << "mk kernel: " << clke_descr(ret) << "\n";

/*
 * ГОТОВИМ ДАННЫЕ
 * */
cl_mem x_mo;
x_mo = NULL;
size_t mem_len = 10000000;

cl_float *x_mem = new cl_float[mem_len];
float *x_cpu_mem = new float[mem_len];
size_t i_mem = 0;
while (i_mem < mem_len)
{
    x_mem[i_mem] = x_cpu_mem[i_mem] = rand();
    i_mem++;
}

cout << "mkbuffers:\n";
//Кидаем данные на видеокарту
x_mo = clCreateBuffer(context, CL_MEM_READ_WRITE, mem_len * sizeof(cl_float),
NULL, &ret);
cout << "\tx: " << clke_descr(ret) << "\n";

cout << "writebuf:\n";
ret = clEnqueueWriteBuffer(command_queue, x_mo, CL_TRUE, 0, mem_len *
sizeof(cl_float), x_mem, 0, NULL, NULL);

```

```

cout << "\tx: " << clke_descr(ret) << "\n";

cout << "set kernargs:\n";
ret = clSetKernelArg(kernel, 0, sizeof(cl_mem), (void *) &x_mo);
cout << "\tx: " << clke_descr(ret) << "\n";

size_t global_work_size[1] = { mem_len };

/* выполнить кернел */
cl_event evt;
cout << "launch gpu: ";
auto start = chrono::system_clock::now();

ret = clEnqueueNDRangeKernel(command_queue, kernel, 1, NULL, global_work_size,
NULL, 0, NULL, &evt);
clWaitForEvents(1, &evt);
clFinish(command_queue);
auto finish = chrono::system_clock::now();

cout << clke_descr(ret) << "\n";
auto dt_gpu = chrono::duration_cast<chrono::milliseconds>(finish-start).count();
/* считать данные из буфера */
ret = clEnqueueReadBuffer(command_queue, x_mo, CL_TRUE, 0, mem_len *
sizeof(cl_float), x_mem, 0, NULL, NULL);
cout<<"launch cpu: ";
start = chrono::system_clock::now();
calculate(x_cpu_mem, mem_len);
finish = chrono::system_clock::now();
cout<<"OK\n";

auto dt_cpu = chrono::duration_cast<chrono::milliseconds>(finish-start).count();
cout<<"Calc time("&<<mem_len<<" pcs):\n";
cout<<"\tCPU: "<<dt_cpu<<" ms\n";
cout<<"\tGPU: "<<dt_gpu<<" ms\n";

return 0;
}

```

Код OpenCL:

```

__kernel void test(__global float* _x)
{
// получаем текущий id.
int id = get_global_id(0);
float x = _x[id];

float ret = sin(x)/x;

_x[id] = ret;
}

```

Код программы «Логарифм»

Код C++ :

```

#include <iostream>
#include <string>
#define CL_USE_DEPRECATED_OPENCL_1_2_APIS
#include <CL/cl.h>
#include <stdio.h>
#include <cmath>
#include <cstdlib>

```

```

#include <chrono>
using namespace std;

std::string clke_descr(cl_int err)
{
    string ret;
    switch (err)
    {
        case CL_SUCCESS:
        {
            ret = "OK";
            break;
        }

        case CL_INVALID_KERNEL:
        {
            ret = "Invalid kernel";
            break;
        }

        case CL_INVALID_KERNEL_ARGS:
        {
            ret = "Invalid kernel args";
            break;
        }

        case CL_INVALID_ARG_INDEX:
        {
            ret = "Invalid arg index";
            break;
        }

        case CL_INVALID_ARG_VALUE:
        {
            ret = "Invalid arg value";
            break;
        }

        case CL_INVALID_MEM_OBJECT:
        {
            ret = "Invalid mem obj";
            break;
        }

        case CL_INVALID_SAMPLER:
        {
            ret = "Invalid sampler";
            break;
        }

        case CL_INVALID_ARG_SIZE:
        {
            ret = "Invalid arg size";
            break;
        }

        case CL_INVALID_PROGRAM:
        {
            ret = "CL_INVALID_PROGRAM";
            break;
        }

        case CL_INVALID_VALUE:

```

```

    {
        ret = "CL_INVALID_VALUE";
        break;
    }
    case CL_INVALID_DEVICE:
    {
        ret = "CL_INVALID_DEVICE";
        break;
    }
    case CL_INVALID_BINARY:
    {
        ret = "CL_INVALID_BINARY";
        break;
    }
    case CL_INVALID_BUILD_OPTIONS:
    {
        ret = "CL_INVALID_BUILD_OPTIONS";
        break;
    }
    case CL_INVALID_OPERATION:
    {
        ret = "CL_INVALID_OPERATION";
        break;
    }
    case CL_COMPILER_NOT_AVAILABLE:
    {
        ret = "CL_COMPILER_NOT_AVAILABLE";
        break;
    }
    case CL_BUILD_PROGRAM_FAILURE:
    {
        ret = "CL_BUILD_PROGRAM_FAILURE";
        break;
    }
    case CL_OUT_OF_HOST_MEMORY:
    {
        ret = "CL_OUT_OF_HOST_MEMORY";
        break;
    }

    default:
    {
        ret = "Unknown err";
        break;
    }
    }
    return ret;
}

int main()
{
    /* получить доступные платформы */
    cl_platform_id platform_id;
    unsigned int ret_num_platforms;

```

```

auto ret = clGetPlatformIDs(1, &platform_id, &ret_num_platforms);
cout << "Num of platforms: " << ret_num_platforms << "\n";

/* получить доступные устройства */
cl_device_id device_id;
unsigned int ret_num_devices;
ret = clGetDeviceIDs(platform_id, CL_DEVICE_TYPE_DEFAULT, 1, &device_id,
                    &ret_num_devices);
cout << "Num of devices: " << ret_num_devices << "\n";
/* создать контекст */
auto context = clCreateContext(NULL, 1, &device_id, NULL, NULL, &ret);

/* создаем команду */
auto command_queue = clCreateCommandQueue(context, device_id, 0, &ret);

cl_program program = NULL;
cl_kernel kernel = NULL;
FILE *fp;

const char fileName[] = "test.cl";
size_t source_size;
string source_str = "";
try
{
    fp = fopen(fileName, "r");
    if (!fp)
    {
        fprintf(stderr, "Failed to load kernel.\n");
        return 1;
    }
    //source_str = (char *)malloc(MAX_SOURCE_SIZE);
    while (!feof(fp))
    {
        char c;
        fread(&c, 1, 1, fp);
        source_str += c;
    }
    fclose(fp);
}
catch (...)
{
    printf("EXCEPTION\n");
}
cout << "Get code: ";
const char * sstrptr = source_str.c_str();
source_size = source_str.length();
//cout<<"-----\n";
//cout<<source_str<<"\n";
//cout<<"-----\n";

/* создать бинарник из кода программы */
cout<<"OK\n";
cout<<"Code ptr: "<<(void *)sstrptr<<"\n";

```

```

program = clCreateProgramWithSource(context, 1, &sstrptr,
                                   (const size_t *) &source_size, &ret);
cout << "create program: " << clke_descr(ret) << "\n";
/* скомпилировать программу */

ret = clBuildProgram(program, 1, &device_id, NULL, NULL, NULL);
cout << "compile: " << clke_descr(ret) << "\n";
if(ret)return -110;
/* создать кернел */
kernel = clCreateKernel(program, "test", &ret);
cout << "mk kernel: " << clke_descr(ret) << "\n";

cl_mem memobj = NULL;
size_t memLenth = 130000000;
cl_int* mem = new cl_int[memLenth];
cl_int* ret_mem = new cl_int[memLenth];
size_t i0 = 0;
while (i0 < memLenth)
{
    mem[i0] = i0;
    i0++;
}
mem[0] = 65536;
cout << "preparing launch\n";
/* создать буфер */
memobj = clCreateBuffer(context, CL_MEM_READ_WRITE,
                        memLenth * sizeof(cl_int), NULL, &ret);

/* записать данные в буфер */
ret = clEnqueueWriteBuffer(command_queue, memobj, CL_TRUE, 0,
                           memLenth * sizeof(cl_int), mem, 0, NULL, NULL);

/* устанавливаем параметр */
ret = clSetKernelArg(kernel, 0, sizeof(cl_mem), (void *) &memobj);

/* устанавливаем параметр */
//ret = clSetKernelArg(kernel, 1, sizeof(cl_mem), (void *) &memobj);

size_t global_work_size[1] =
{ memLenth };
cl_event evt;
cout << "launch gpu: ";
auto start = chrono::system_clock::now();

/* выполнить кернел */
ret = clEnqueueNDRangeKernel(command_queue, kernel, 1, NULL,
global_work_size, NULL, 0, NULL, &evt);
cout<<((ret==CL_SUCCESS)?("OK\n")):("Not OK\n"));
if(ret!=CL_SUCCESS)return -15;
clWaitForEvents(1, &evt);
cout<<"EVTs\n";
clFinish(command_queue);
cout<<"Finish\n";
auto finish = chrono::system_clock::now();

//cout << clke_descr(ret) << "\n";

```

```

        auto dt_gpu = chrono::duration_cast<chrono::milliseconds>(finish -
start).count();

    cout<<"calc time ("<<memLenth<<"):\n";
    cout <<"\tGPU: "<<dt_gpu<<" ms\n";
    /* выполнить кернел */

    /* считать данные из буфера */
    ret = clEnqueueReadBuffer(command_queue, memobj, CL_TRUE, 0,
                             memLenth * sizeof(float), ret_mem, 0, NULL, NULL);
    //size_t i = 0;

    /*while (i < memLenth)
    {
        if (ret_mem[i] >= 0)
        {
            cout << "log2(" << mem[i] << ") = " << ret_mem[i] << "\n";
        }
        else
        {
            cout << "Value " << mem[i] << " isn\'t a 2^k\n";
        }
        i++;
    }*/
    return 0;
}

```

Код OpenCL:

```

__kernel void test(__global int* message)
{
    // получаем текущий id.
    int gid = get_global_id(0);
    int gg = gid;
    int gv = message[gg];
    int rv = 0;
    if(gv<1){
        message[gg] = -1;
        return;
    }
    if(gv==1){
        message[gg]=0;
        return;
    }
    //message[gg] = 0;

    while(gv>1){
        if(gv%2){
            rv = -1;
            break;
        }
        gv /= 2;
        rv++;
    }
    message[gg] = rv;
    //message[gid] = gid;
}

```

Код программы «Геометрия»

Код C++ :

```
#include <iostream>
#include <string>
#define CL_USE_DEPRECATED_OPENCL_1_2_APIS
#include <CL/cl.h>
#include <stdio.h>
#include <cmath>
#include <cstdlib>
#include <chrono>

using namespace std;

int main()
{
    /* получить доступные платформы */
    cl_platform_id platform_id;
    unsigned int ret_num_platforms;
    cl_int ret = clGetPlatformIDs(1, &platform_id, &ret_num_platforms);
    cout << "Num of platforms: " << ret_num_platforms << "\n";

    /* получить доступные устройства */
    cl_device_id device_id;
    unsigned int ret_num_devices;
    ret = clGetDeviceIDs(platform_id, CL_DEVICE_TYPE_DEFAULT, 1, &device_id,
                        &ret_num_devices);
    cout << "Num of devices: " << ret_num_devices << "\n";
    /* создать контекст */
    cl_context context = clCreateContext(NULL, 1, &device_id, NULL, NULL, &ret);
    cout<<"Create context: "<<clke_descr(ret)<<"\n";
    /* создаем команду */
    cl_command_queue command_queue = clCreateCommandQueue(context, device_id, 0,
                                                         &ret);
    cout<<"Create queue: "<<clke_descr(ret)<<"\n";

    cl_program program = NULL;
    cl_kernel kernel = NULL;
    FILE *fp;
    const char fileName[] = "test.cl";
    size_t source_size;
    string source_str = "";
    try
    {
        fp = fopen(fileName, "r");
        if (!fp)
        {
            fprintf(stderr, "Failed to load kernel.\n");
            return 1;
        }
        //source_str = (char *)malloc(MAX_SOURCE_SIZE);
        while (!feof(fp))
        {
            char c;
            fread(&c, 1, 1, fp);
```



```

        source_str += c;

    }
    fclose(fp);
}
catch (...)
{
    printf("EXCEPTION\n");
}
cout << "Get code: ";
const char * sstrptr = source_str.c_str();
source_size = source_str.length();

/* создать бинарник из кода программы */
cout<<"OK\n";
cout<<"Code ptr: " <<(void *)sstrptr<<"\n";
program = clCreateProgramWithSource(context, 1, &sstrptr,
                                   (const size_t *) &source_size, &ret);
cout << "create program: " << clke_descr(ret) << "\n";
/* скомпилировать программу */

ret = clBuildProgram(program, 1, &device_id, NULL, NULL, NULL);
cout << "compile: " << clke_descr(ret) << "\n";
if(ret)return -110;
/* создать кернел */
kernel = clCreateKernel(program, "test", &ret);
cout << "mk kernel: " << clke_descr(ret) << "\n";

/*
 * Готовим векторы, выделяем память
 * */
cl_mem u_mo, p_mo, a_mo, b_mo, c_mo;
u_mo = p_mo = a_mo = b_mo = c_mo = NULL;
size_t mem_len = 10000000;

cl_float3 *vec_u_mem = new cl_float3[mem_len];
cl_float3 *dot_p_mem = new cl_float3[mem_len];
cl_float3 *dot_a_mem = new cl_float3[mem_len];
cl_float3 *dot_b_mem = new cl_float3[mem_len];
cl_float3 *dot_c_mem = new cl_float3[mem_len];

size_t i_mem = 0;
while (i_mem < mem_len)
{
    vec_u_mem[i_mem].s[0] = rand();
    vec_u_mem[i_mem].s[1] = rand();
    vec_u_mem[i_mem].s[2] = rand();
    dot_p_mem[i_mem].s[0] = rand();
    dot_p_mem[i_mem].s[1] = rand();
    dot_p_mem[i_mem].s[2] = rand();
    dot_a_mem[i_mem].s[0] = rand();
    dot_a_mem[i_mem].s[1] = rand();
    dot_a_mem[i_mem].s[2] = rand();
    dot_b_mem[i_mem].s[0] = rand();
    dot_b_mem[i_mem].s[1] = rand();
    dot_b_mem[i_mem].s[2] = rand();
}

```

```

        dot_c_mem[i_mem].s[0] = rand();
        dot_c_mem[i_mem].s[1] = rand();
        dot_c_mem[i_mem].s[2] = rand();

        i_mem++;
    }

    cout << "mkbuffers:\n";
    //Кидаем данные на видеокарту
    u_mo = clCreateBuffer(context, CL_MEM_READ_WRITE,
                           mem_len * sizeof(cl_float3), NULL, &ret);
    cout << "u: " << clke_descr(ret) << "\n";
    p_mo = clCreateBuffer(context, CL_MEM_READ_WRITE,
                           mem_len * sizeof(cl_float3), NULL, &ret);
    cout << "p: " << clke_descr(ret) << "\n";
    a_mo = clCreateBuffer(context, CL_MEM_READ_WRITE,
                           mem_len * sizeof(cl_float3), NULL, &ret);
    cout << "a: " << clke_descr(ret) << "\n";
    b_mo = clCreateBuffer(context, CL_MEM_READ_WRITE,
                           mem_len * sizeof(cl_float3), NULL, &ret);
    cout << "b: " << clke_descr(ret) << "\n";
    c_mo = clCreateBuffer(context, CL_MEM_READ_WRITE,
                           mem_len * sizeof(cl_float3), NULL, &ret);
    cout << "c: " << clke_descr(ret) << "\n";
    cout << "writebuf:\n";
    ret = clEnqueueWriteBuffer(command_queue, u_mo, CL_TRUE, 0,
                               mem_len * sizeof(cl_float3), vec_u_mem, 0, NULL,
    NULL);
    cout << "u: " << clke_descr(ret) << "\n";
    ret = clEnqueueWriteBuffer(command_queue, p_mo, CL_TRUE, 0,
                               mem_len * sizeof(cl_float3), dot_p_mem, 0, NULL,
    NULL);
    cout << "p: " << clke_descr(ret) << "\n";
    ret = clEnqueueWriteBuffer(command_queue, a_mo, CL_TRUE, 0,
                               mem_len * sizeof(cl_float3), dot_a_mem, 0, NULL,
    NULL);
    cout << "a: " << clke_descr(ret) << "\n";
    ret = clEnqueueWriteBuffer(command_queue, b_mo, CL_TRUE, 0,
                               mem_len * sizeof(cl_float3), dot_b_mem, 0, NULL,
    NULL);
    cout << "b: " << clke_descr(ret) << "\n";
    ret = clEnqueueWriteBuffer(command_queue, c_mo, CL_TRUE, 0,
                               mem_len * sizeof(cl_float3), dot_c_mem, 0, NULL,
    NULL);
    cout << "c: " << clke_descr(ret) << "\n";

    cout << "set kernargs:\n";
    ret = clSetKernelArg(kernel, 0, sizeof(cl_mem), (void *) &u_mo);
    cout << "u: " << clke_descr(ret) << "\n";

    ret = clSetKernelArg(kernel, 1, sizeof(cl_mem), (void *) &p_mo);
    cout << "p: " << clke_descr(ret) << "\n";

    ret = clSetKernelArg(kernel, 2, sizeof(cl_mem), (void *) &a_mo);
    cout << "a: " << clke_descr(ret) << "\n";

```

```

ret = clSetKernelArg(kernel, 3, sizeof(cl_mem), (void *) &b_mo);
cout << "b: " << clke_descr(ret) << "\n";

ret = clSetKernelArg(kernel, 4, sizeof(cl_mem), (void *) &c_mo);
cout << "c: " << clke_descr(ret) << "\n";

size_t global_work_size[1] =
{ mem_len };
cl_event evt;
cout << "launch gpu: ";
auto start = chrono::system_clock::now();

/* выполнить kernel */
ret = clEnqueueNDRangeKernel(command_queue, kernel, 1, NULL,
global_work_size, NULL, 0, NULL, &evt);
clWaitForEvents(1, &evt);
clFinish(command_queue);
auto finish = chrono::system_clock::now();

cout << clke_descr(ret) << "\n";
auto dt_gpu = chrono::duration_cast<chrono::milliseconds>(finish -
start).count();

cout<<"calc time ("<<mem_len<<"):\n";
cout <<"\tGPU: "<<dt_gpu<<" ms\n";

/* считать данные из буфера */
ret = clEnqueueReadBuffer(command_queue, u_mo, CL_TRUE, 0,
mem_len * sizeof(cl_float3), vec_u_mem, 0, NULL,
NULL);
ret = clEnqueueReadBuffer(command_queue, p_mo, CL_TRUE, 0,
mem_len * sizeof(cl_float3), dot_p_mem, 0, NULL,
NULL);
ret = clEnqueueReadBuffer(command_queue, a_mo, CL_TRUE, 0,
mem_len * sizeof(cl_float3), dot_a_mem, 0, NULL,
NULL);
ret = clEnqueueReadBuffer(command_queue, b_mo, CL_TRUE, 0,
mem_len * sizeof(cl_float3), dot_b_mem, 0, NULL,
NULL);
ret = clEnqueueReadBuffer(command_queue, c_mo, CL_TRUE, 0,
mem_len * sizeof(cl_float3), dot_c_mem, 0, NULL,
NULL);

return 0;
}

```

Код OpenCL:

```

//скалярное произведение векторов
float mscalar (float3 a, float3 b)
{

    return ((a.x*b.x) + (a.y*b.y) + (a.z*b.z));
}

```

```

//векторное произведение векторов
float3 mvector(float3 a, float3 b)
{
    float i, j, k;
    i = a.y*b.z - a.z*b.y;
    j = a.z*b.x - a.x*b.z;
    k = a.x*b.y - a.y*b.x;
    float3 v;
    v.x = i;
    v.y = j;
    v.z = k;
    return v;
}

//Модуль вектора
float modvector(float3 a)
{
    return sqrt(a.x*a.x+a.y*a.y+a.z*a.z);
}

//Проекция вектора a на направление вектора b
float prvector(float3 a, float3 b)
{
    float retval;
    float modb = sqrt(b.x*b.x+b.y*b.y+b.z*b.z);
    retval = mscalar(a,b)/modb;
    return retval;
}

float3 normvector(float3 a)
{
    float3 ret;
    float len = modvector(a);
    ret.x = a.x/len;
    ret.y = a.y/len;
    ret.z = a.z/len;
    return ret;
}

__kernel void test(__global float3* u, __global float3* p, __global float3* a,
__global float3* b, __global float3* c)
{
    // получаем текущий id.
    int id = get_global_id(0);
    float3 uv = u[id];
    float3 pv = p[id];

    float3 av = a[id];
    float3 bv = b[id];
    float3 cv = c[id];

    a[id] = u[id];
    uv = normvector(uv);

    //Бектор ab
    float3 abv = bv-av;
    //Бектор ac

```

```

float3 acv = cv - av;
//Нормаль к плоскости
float3 nv = normvector(mvector(abv, acv));
//Скалярное произведение нормали и вектора луча
float un_s = mscalar(nv, uv);
//Случай, когда луч параллелен плоскости
if(un_s==0)
{

    uv.x = 0;
    uv.y = 1;
    uv.z = 0;
    u[id] = uv;

    p[id].x = un_s;

    return;
}

//Число d(расстояние от плоскости до начала координат)
//Определяется проекцией любой точки, принадлежащей плоскости, на вектор нормали.
float d_s = prvector(av, -nv);
//Число pdd - расстояние от точки до плоскости вдоль обратного вектора нормали
float pdz_s = prvector(pv, -nv);
float pdd_s = pdz_s-d_s;
float pv_s = mscalar(pv,nv);
a[id].x = pv_s;
float alpha_s = (-d_s - pv_s)/un_s;
//Случай, когда луч направлен от плоскости
if(alpha_s<=0)
{
    uv.x = 0;
    uv.y = 0;
    uv.z = 1;
    u[id] = uv;
    return;
}
a[id].y = alpha_s;
//Точка пересечения луча и плоскости
float3 auv;
auv.x = alpha_s*uv.x;
auv.y = alpha_s*uv.y;
auv.z = alpha_s*uv.z;
float3 i_v;// = pv + auv;
i_v.x = pv.x+auv.x;
i_v.y = pv.y+auv.y;
i_v.z = pv.z+auv.z;
float3 U,V,F, T;
float Um, Vm, Fm, Tm;
T = mvector(bv-av, cv-av);
U = mvector(i_v-av, i_v-bv);
V = mvector(i_v-bv, i_v-cv);
F = mvector(i_v-av, i_v-cv);

//Модули площадей параллелограммов
Tm = modvector(T);
Um = modvector(U);
Vm = modvector(V);
Fm = modvector(F);

```

```

//Точка пересечения с плоскостью за пределами треугольника
if(((Um+Vm)>=Tm)||((Fm+Vm)>=Tm)||((Um+Fm)>=Tm))
{
    uv.x = 0;
    uv.y = 1;
    uv.z = 1;
    u[id] = uv;
    //i_v.x = Um;
    //i_v.y = Vm;
    //i_v.z = Tm;
    p[id].x = d_s;
    p[id].y = alpha_s;
    return;
}
//Точка пересечения внутри треугольника;
uv.x = uv.y = uv.z = 1;
u[id] = uv;
p[id] = i_v;
//a[id] = p[id];

}

```