

OpenCL

Краткое введение

Что есть OpenCL?

- **Open Computing Language** – это фреймворк, предназначенный для параллельного программирования на различных типах вычислительных устройств (таких как CPU, GPU и других), с собственным языком программирования, основанным на C99.
- Изначально основан Apple.
- Свободен и бесплатен.

Установка: проверка поддержки

Есть несколько способов проверки. Рассмотрим на примере GPU:

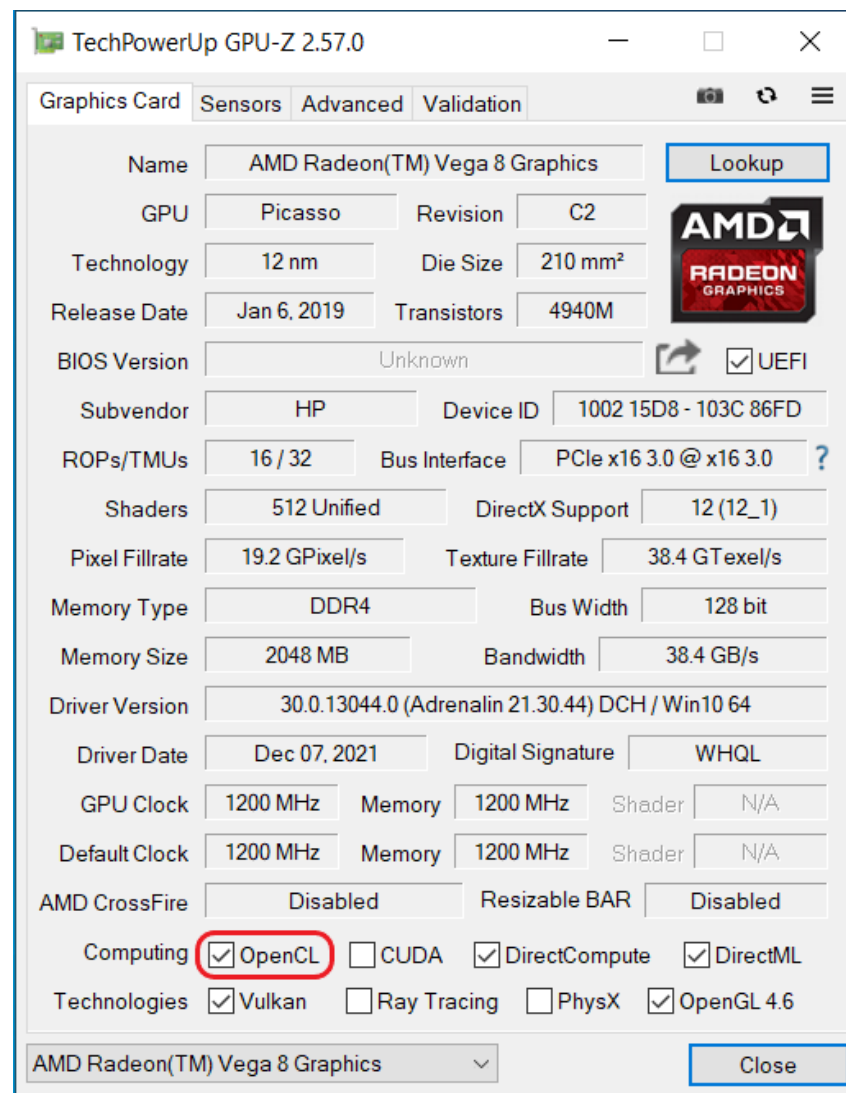
- (Универсальный) Погуглить по идентификатору устройства.
- (ОС Windows) Зайти в драйверы устройства и найти *opencl.dll*.
- (ОС на базе Linux) `$ clinfo`
- (Рекомендуемый, ОС Windows)
Скачать *TechPowerUp GPU-Z* и посмотреть, отмечено ли на устройстве галочка рядом с *OpenCL*.

Установка: GPU-Z

```
$ winget install TechPowerUp.GPU-Z
```

```
Labour@DESKTOP-JK73HE5 MINGW64 ~  
$ winget search gpu-z  
Имя                ИД                Версия Совпадение  Источник  
-----  
TechPowerUp GPU-Z  TechPowerUp.GPU-Z  2.57.0 Moniker: gpu-z winget  
  
Labour@DESKTOP-JK73HE5 MINGW64 ~  
$ winget install TechPowerUp.GPU-Z  
Найдено TechPowerUp GPU-Z [TechPowerUp.GPU-Z] Версия 2.57.0  
Лицензия на это приложение предоставлена вам владельцем.  
Корпорация Майкрософт не несет ответственность за сторонние пакеты и не предоставляет для них никакие лицензии.  
Скачивание https://us2-dl.techpowerup.com/files/GPU-Z.2.57.0.exe  
[REDACTED] 9.64 MB / 9.64 MB  
Хэш установщика успешно проверен  
Запуск установки пакета...  
Успешно установлено
```

Установка: GPU-Z



Установка: драйвера

- Иногда для работы OpenCL требуется получить полный пакет драйверов GPU.
- AMD:
<https://www.amd.com/en/support>

DOWNLOAD WINDOWS DRIVERS



Установка: SDK

- Инструкция от разработчиков: <https://github.com/KhronosGroup/OpenCL-Guide?tab=readme-ov-file#the-opencl-sdk>
- (Linux) В зависимости от дистрибутива придётся установить devel пакеты, по типу *OpenCL-Headers*, *libclc*, *OpenCL-CLHPP*, *mesa-opencl* или *nvidia-opencl*.
- (Windows)
 - Используя пакетный менеджер vsrpg от Visual Studio.
 - Либо же скачать подготовленный SDK:
<https://github.com/GPUOpen-LibrariesAndSDKs/OCL-SDK/releases/latest>

OCL_SDK_Light

Latest



BenjaminCoquelle released this Jun 27, 2017

· 1 commit to master since this release



1.0

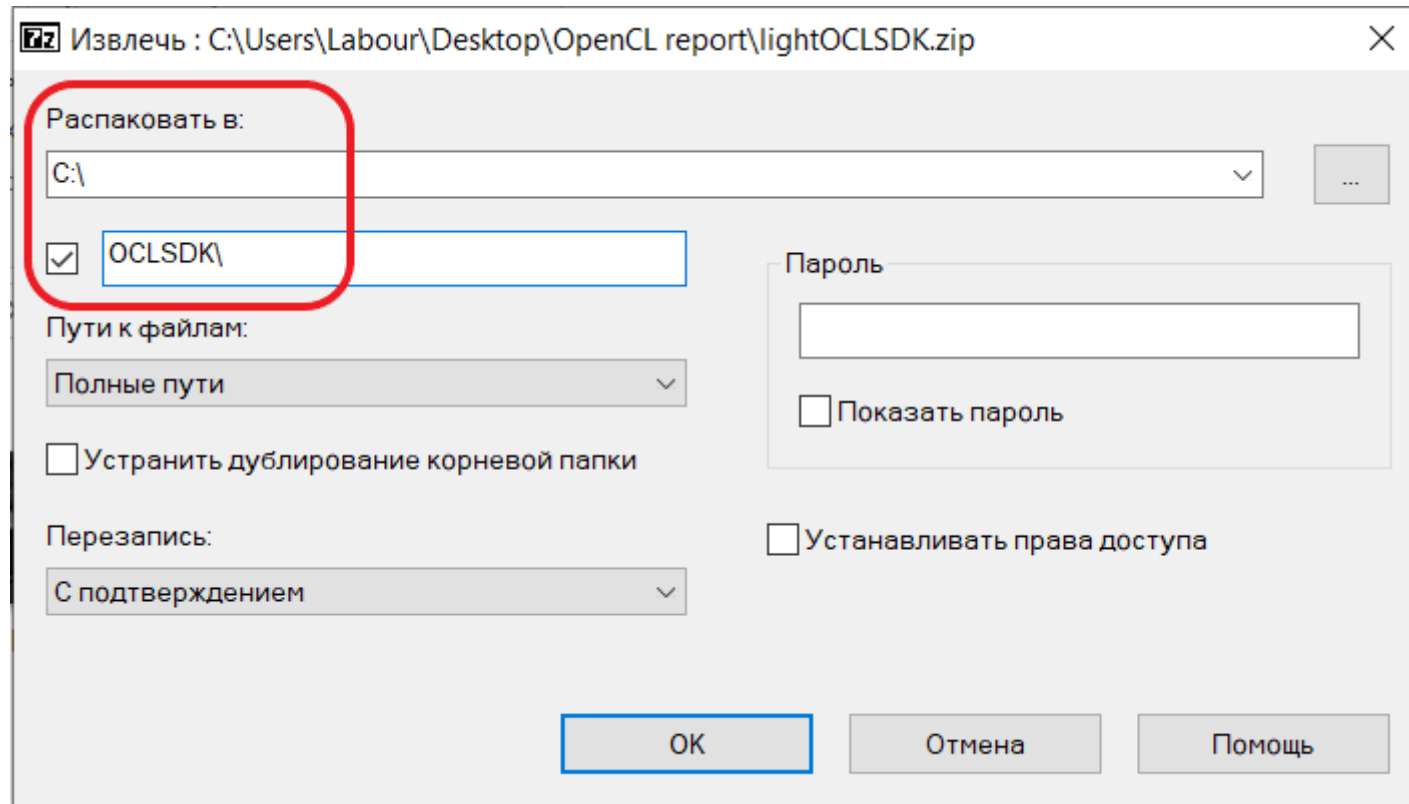


468cd3d

[lightOCLSDK.zip](#)

Установка: SDK

- Необходимо запомнить путь распаковки, по надобности можно добавить в *%PATH%*.



Установка: MinGW

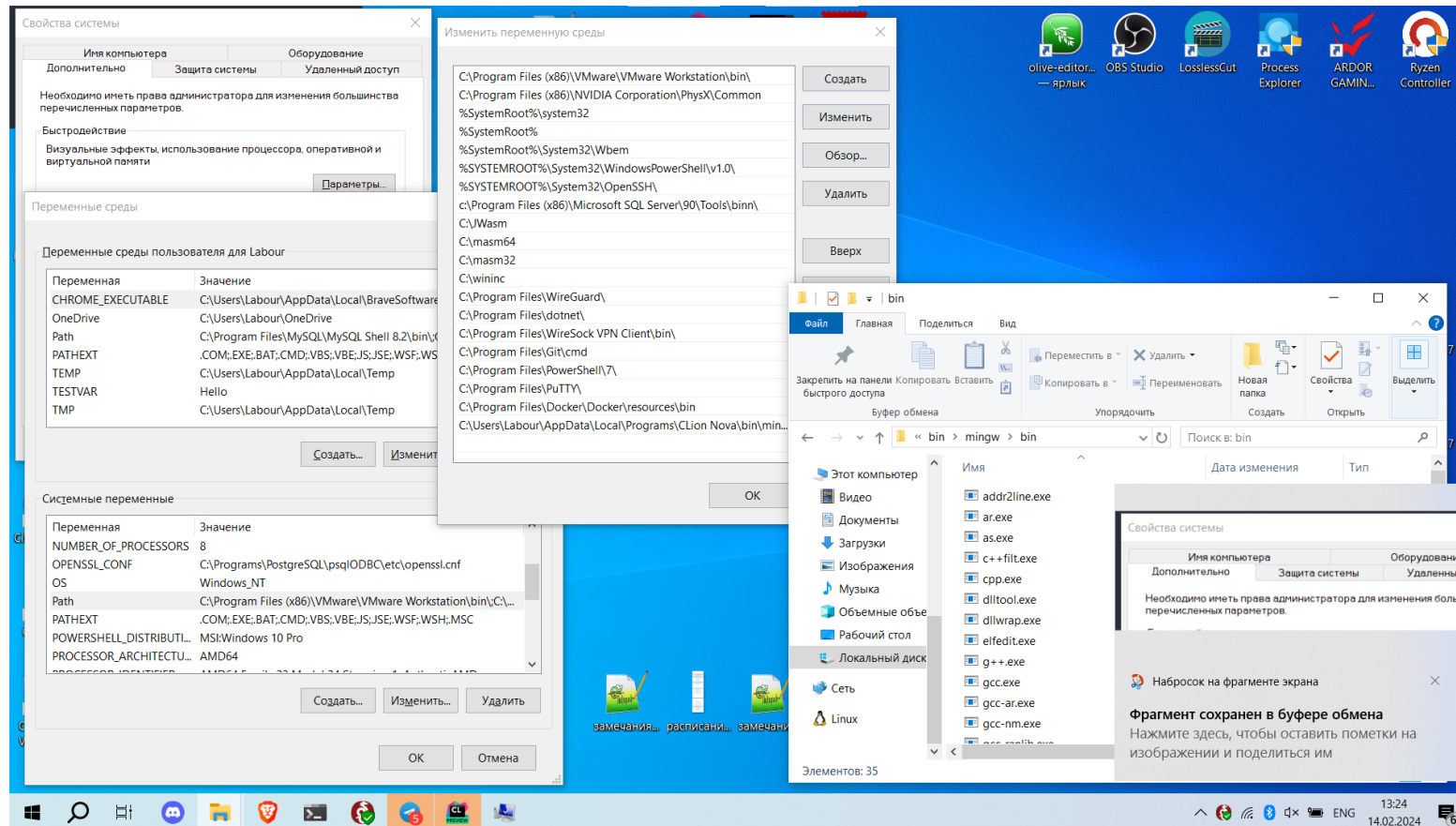
- Потребуется инструментарий разработки, в данном случае я буду использовать MinGW: <https://winlibs.com/>.

UCRT runtime

- GCC 13.2.0 (with **POSIX** threads) + LLVM/Clang/LLD/LLDB 17.0.6 + MinGW-w64 11.0.1 (UCRT) - release 5 **(LATEST)**
 - Win32: [7-Zip archive*](#) | [Zip archive](#) - without LLVM/Clang/LLD/LLDB: [7-Zip archive*](#) | [Zip archive](#)
 - Win64: [7-Zip archive*](#) | [Zip archive](#) - without LLVM/Clang/LLD/LLDB: [7-Zip archive*](#) | [Zip archive](#)
- GCC 13.2.0 (with **MCF** threads) + MinGW-w64 11.0.1 (UCRT) - release 3 **(LATEST)**
 - Win32 (without LLVM/Clang/LLD/LLDB): [7-Zip archive*](#) | [Zip archive](#)
 - Win64 (without LLVM/Clang/LLD/LLDB): [7-Zip archive*](#) | [Zip archive](#)
- GCC 13.1.0 (with **POSIX** threads) + LLVM/Clang/LLD/LLDB 16.0.5 + MinGW-w64 11.0.0 (UCRT) - release 5
 - Win32: [7-Zip archive*](#) | [Zip archive](#) - without LLVM/Clang/LLD/LLDB: [7-Zip archive*](#) | [Zip archive](#)
 - Win64: [7-Zip archive*](#) | [Zip archive](#) - without LLVM/Clang/LLD/LLDB: [7-Zip archive*](#) | [Zip archive](#)
- GCC 13.1.0 (with **MCF** threads) + MinGW-w64 11.0.0 (UCRT) - release 5
 - Win32 (without LLVM/Clang/LLD/LLDB): [7-Zip archive*](#) | [Zip archive](#)
 - Win64 (without LLVM/Clang/LLD/LLDB): [7-Zip archive*](#) | [Zip archive](#)

Установка: MinGW

- Не забудьте добавить в %PATH% после распаковки по пути <...>/mingw/bin



Проверка

- Код доступен по ссылке:

https://github.com/stdStudent/ocl_example/blob/f994354873900105d2d4b4e8c397e467705204b9/main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <CL/cl.h>

int main() {
    // Get the number of platforms
    cl_uint numPlatforms;
    clGetPlatformIDs(0, NULL, &numPlatforms);

    if (numPlatforms == 0) {
        printf("No OpenCL platforms found.\n");
        return 1;
    }

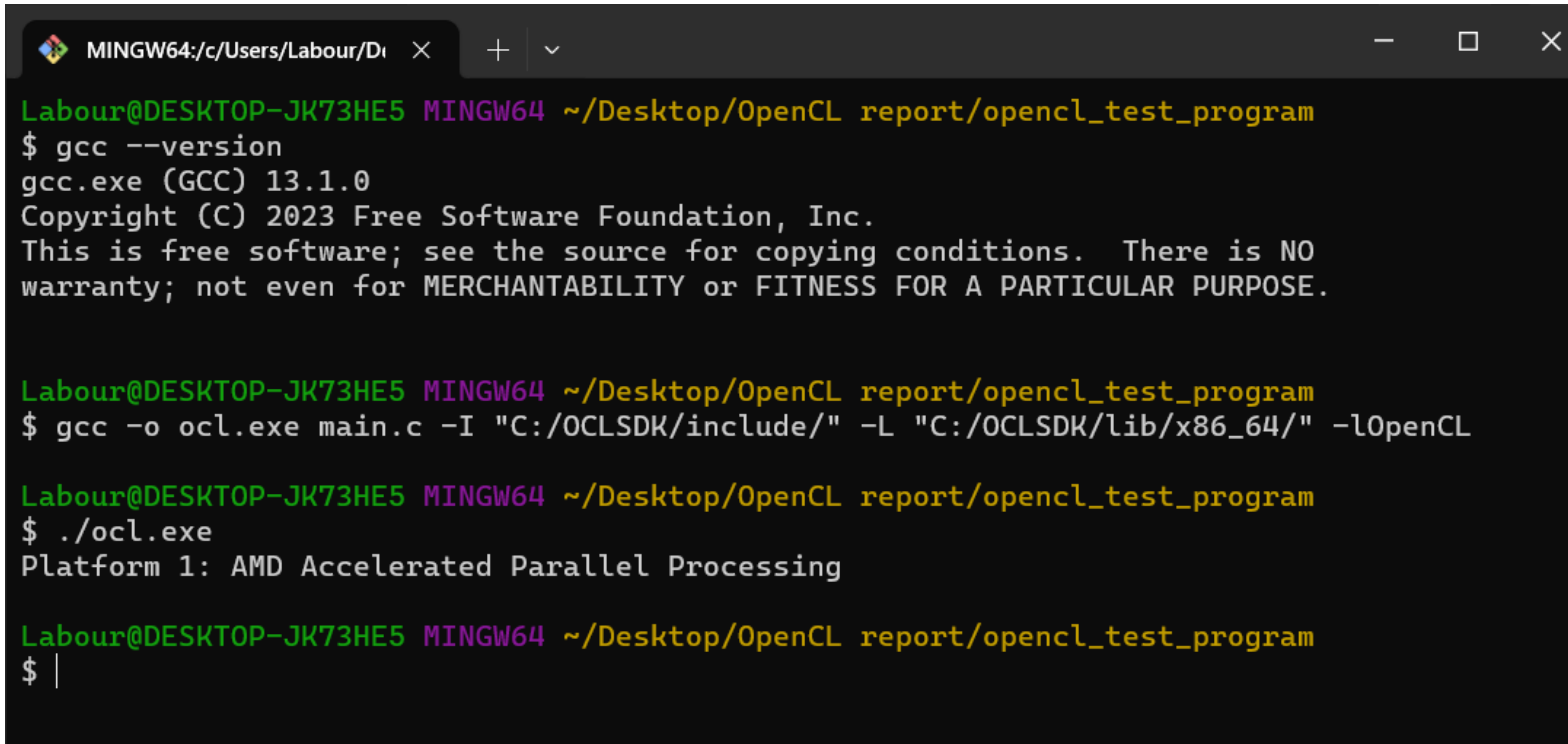
    // Get the platform IDs
    cl_platform_id* platforms = malloc(sizeof(cl_platform_id) * numPlatforms);
    clGetPlatformIDs(numPlatforms, platforms, NULL);

    // Print the platform names
    for (cl_uint i = 0; i < numPlatforms; ++i) {
        char buffer[1024];
        clGetPlatformInfo(platforms[i], CL_PLATFORM_NAME, sizeof(buffer), buffer, NULL);
        printf("Platform %d: %s\n", i + 1, buffer);
    }

    free(platforms);
    return 0;
}
```

Проверка

- `-o ocl.exe` указывает название выходного файла
- `-I "C:/OCLSDK/include/"` указывает на папку с заголовочными файлами
- `-L "C:/OCLSDK/lib/x86_64/"` указывает на папку с библиотекой для линковки
- `-lOpenCL` линкует выходной бинарник с OpenCL



```
MINGW64:/c/Users/Labour/Di x + v
Labour@DESKTOP-JK73HE5 MINGW64 ~/Desktop/OpenCL report/openc1_test_program
$ gcc --version
gcc.exe (GCC) 13.1.0
Copyright (C) 2023 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Labour@DESKTOP-JK73HE5 MINGW64 ~/Desktop/OpenCL report/openc1_test_program
$ gcc -o ocl.exe main.c -I "C:/OCLSDK/include/" -L "C:/OCLSDK/lib/x86_64/" -lOpenCL

Labour@DESKTOP-JK73HE5 MINGW64 ~/Desktop/OpenCL report/openc1_test_program
$ ./ocl.exe
Platform 1: AMD Accelerated Parallel Processing

Labour@DESKTOP-JK73HE5 MINGW64 ~/Desktop/OpenCL report/openc1_test_program
$ |
```

Настройка CMake проекта

- Код доступен по ссылке:

https://github.com/stdStudent/ocl_example/blob/master/CMakeLists.txt

```
cmake_minimum_required(VERSION 3.28)|
project(ocl_example C)

set(CMAKE_C_STANDARD 17)

add_executable(ocl_example main.c)

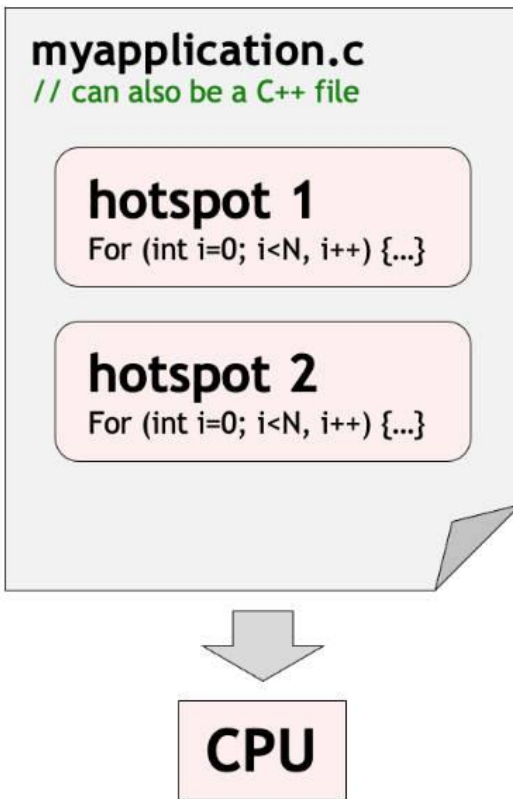
# Set the include directories (-I "C:/OCLSDK/include/")
target_include_directories(ocl_example PRIVATE "C:/OCLSDK/include/")

# Specify the link directories (-L "C:/OCLSDK/lib/x86_64/")
link_directories("C:/OCLSDK/lib/x86_64/")

# Link the OpenCL library (-lOpenCL)
target_link_libraries(ocl_example "C:/OCLSDK/lib/x86_64/OpenCL.lib")
```

Традиционная парадигма программирования в сравнении с OpenCL

C/C++ Programming



OpenCL Programming

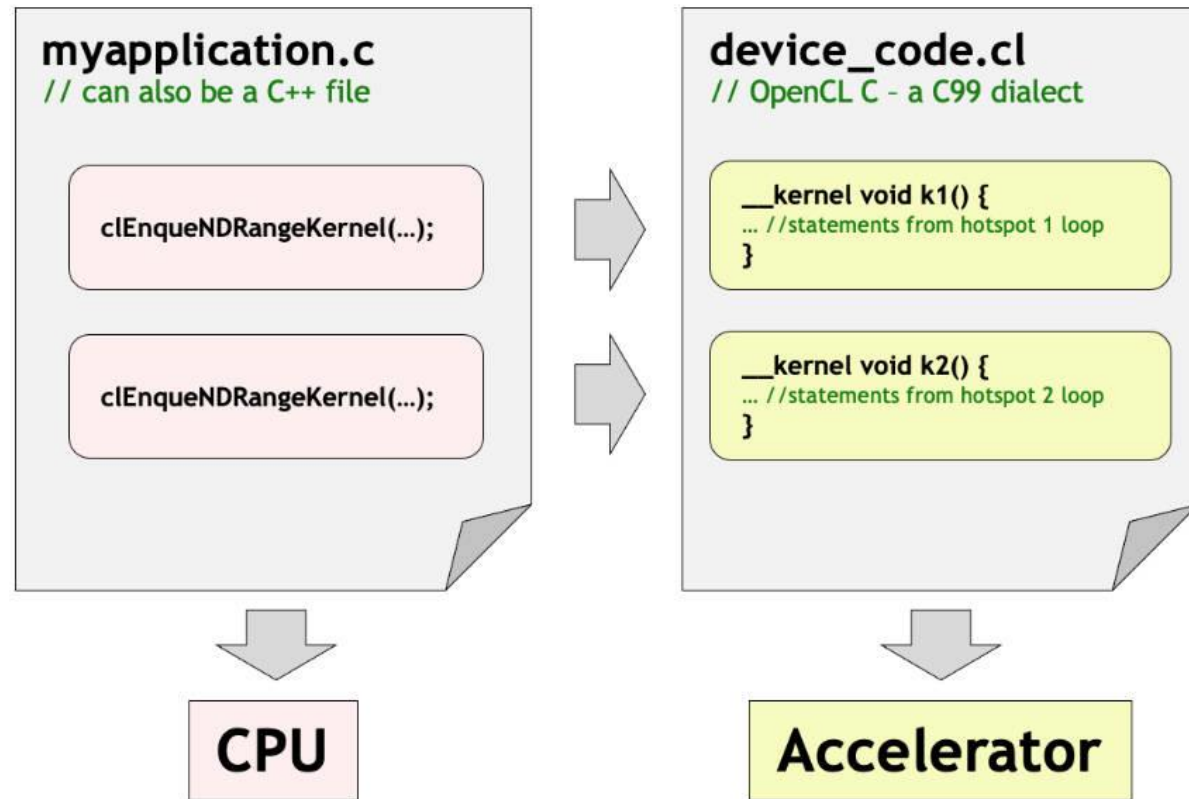
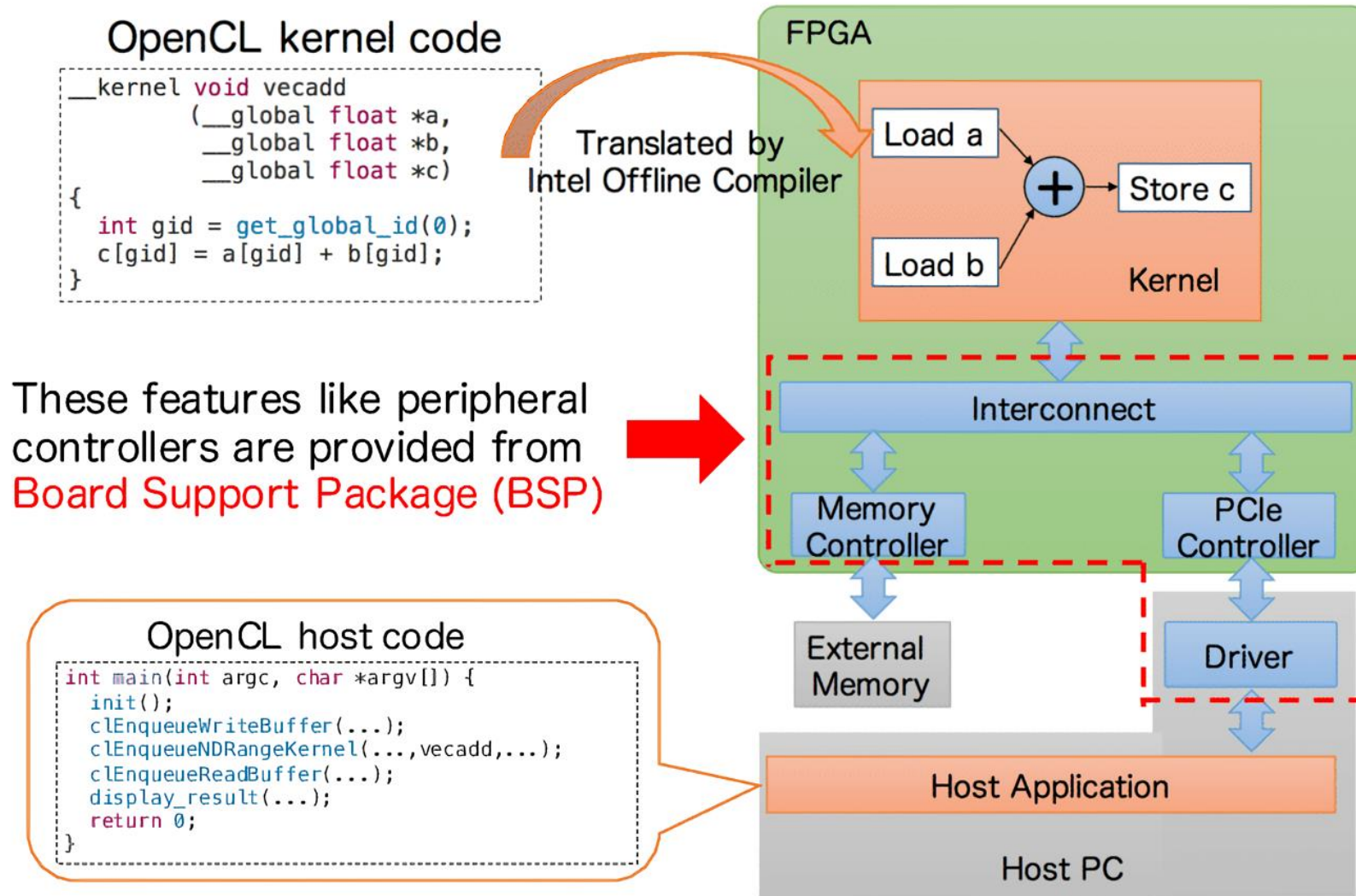


Схема работы программы



OpenCL код

```
__kernel void mul_kernel(  
    __global double *matrix1_in,  
    __global double *matrix2_in,  
    __global double *matrix_out,  
    int size  
) {  
    // The kernel allows for a more parallelizable implementation,  
    // where each entry of the resulting matrix is calculated by a single kernel  
    int id = get_global_id(0);  
    double value = 0;  
    int row, col;  
  
    // The kernel assigns a row of the first operand and a column of the second operand that is computed  
    // (multiplying its entries and adding them together) to achieve the final result for that specific entry  
    // in the resulting matrix  
    row = id / size;  
    col = id % size;  
    for (int i = 0; i < size; ++i)  
        value += matrix2_in[(i * size) + col] * matrix1_in[(row * size) + i];  
  
    matrix_out[id] = value;  
}
```


OpenCL код

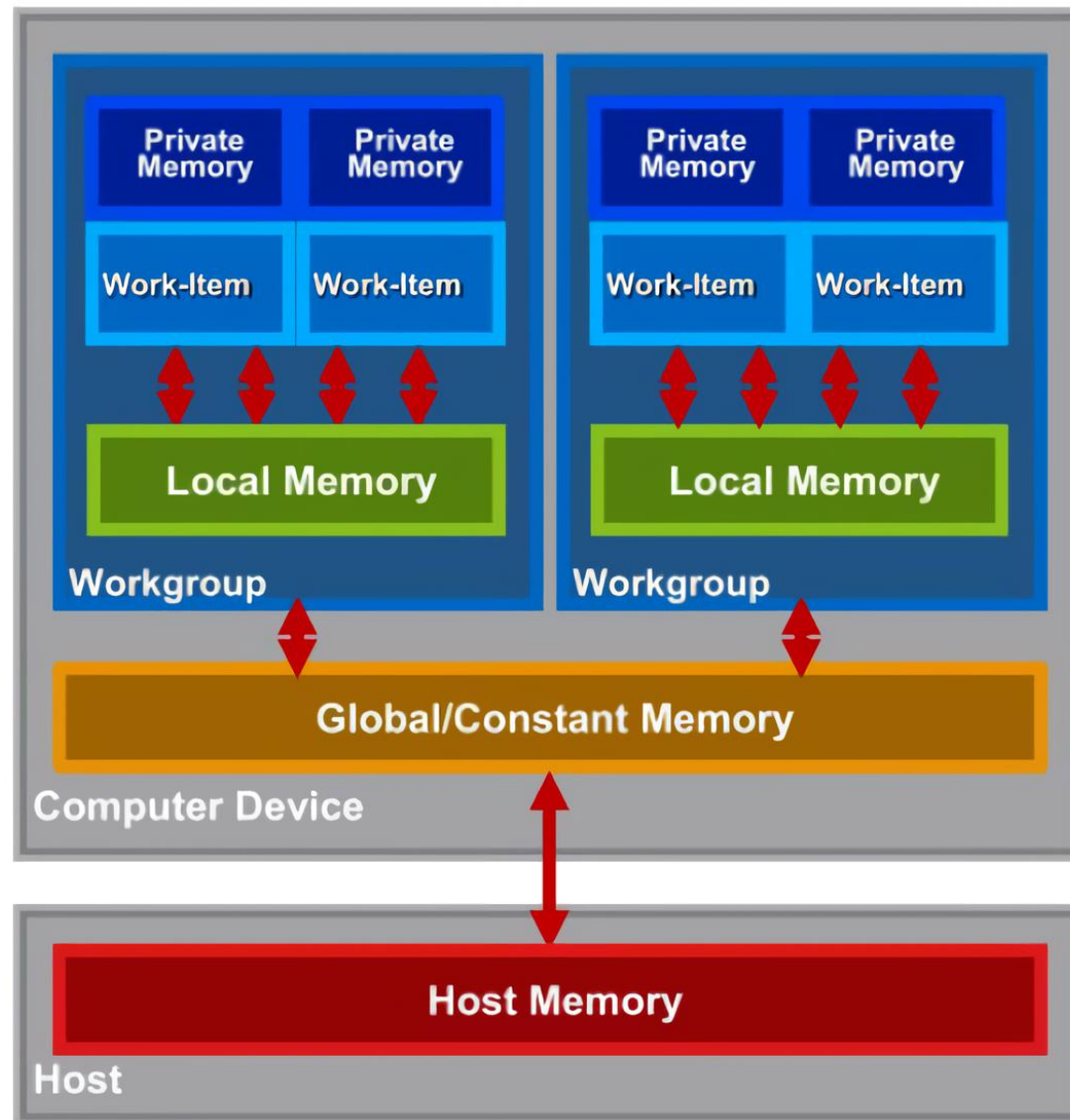
- `__kernel`: Это ключевое слово, которое определяет функцию ядра OpenCL. Ядро — это функция, которая будет выполняться на устройстве параллельно.
- `__global`: Это квалификатор доступа к памяти, который указывает, что переменные `matrix1_in`, `matrix2_in`, и `matrix_out` являются глобальными и доступны для всех потоков ядра.
- `get_global_id(0)`: Это функция OpenCL, которая возвращает уникальный идентификатор потока внутри глобальной рабочей группы. В данном случае используется один размер (0), что означает одномерную рабочую группу.

Вместо того чтобы использовать двумерное пространство индексов (одно измерение для строк, другое для столбцов), код использует одномерное пространство и вычисляет соответствующую строку и столбец с помощью деления и взятия остатка от деления.

- `get_global_id(0)` возвращает текущий индекс в глобальном пространстве индексов. Поскольку используется одномерное пространство, это будет числом от 0 до `size*size - 1`.
- `row = id / size` вычисляет номер строки, которую должен обрабатывать этот рабочий элемент. Деление на `size` эффективно “отбрасывает” остаток, оставляя только целую часть результата.
- `col = id % size` вычисляет номер столбца, который должен обрабатывать этот рабочий элемент.

Таким образом, каждый рабочий элемент получает уникальную пару `(row, col)`, которую он использует для вычисления соответствующего элемента в результирующей матрице.

Модель памяти OpenCL



Написание хостовой программы

- Обнаружение и инициализация платформ
- Обнаружение и инициализация устройств
- Создание контекста
- Создание очереди команд
- Создание буферов устройств и копирование данных в них
- Создание и компиляция OpenCL программы
- Создание ядра
- Установка аргументов ядра
- Настройка структуры рабочих элементов
- Освобождение ресурсов OpenCL

Шаг 1: Обнаружение и инициализация платформ

- `clGetPlatformIDs` вызывается дважды: сначала для получения числа доступных платформ (`numPlatforms`), а затем для заполнения массива `platforms` идентификаторами этих платформ.

```
cl_uint numPlatforms = 0;
cl_platform_id *platforms = NULL;

// Retrieve the number of platforms
status = clGetPlatformIDs(0, NULL, &numPlatforms);
CHECK(status);

// Allocate enough space for each platform
platforms = (cl_platform_id *) malloc(sizeof(cl_platform_id) * numPlatforms);

// Fill in platforms with clGetPlatformIDs()
status = clGetPlatformIDs(numPlatforms, platforms, NULL);
CHECK(status);
```

Шаг 2: Обнаружение и инициализация устройств

```
cl_uint numDevices = 0;
cl_device_id *devices = NULL;

// Use clGetDeviceIDs() to retrieve the number of devices present
status = clGetDeviceIDs(
    platforms[0],
    CL_DEVICE_TYPE_GPU,
    0,
    NULL,
    &numDevices
);
CHECK(status);

// Allocate enough space for each device
devices = (cl_device_id *) malloc(sizeof(cl_device_id) * numDevices);

// Fill in devices with clGetDeviceIDs()
status |= clGetDeviceIDs(
    platforms[0],
    CL_DEVICE_TYPE_GPU,
    numDevices,
    devices,
    NULL
);
CHECK(status);

// Select the device which will be used
const int device_id = 0;
```

- **clGetDeviceIDs** также вызывается дважды: сначала для получения числа доступных устройств (**numDevices**) для первой платформы, а затем для заполнения массива **devices** идентификаторами этих устройств.
- Затем выбирается устройство, которое будет использоваться (в данном случае, это первое устройство).

Шаг 3: Создание контекста

- Контекст в OpenCL используется для управления ресурсами, такими как устройства, очереди команд и объекты памяти, которые используются в программ

```
cl_context context = NULL;

// Create a context using clCreateContext() and
// associate it with the devices
context = clCreateContext(
    NULL,
    1,
    &devices[device_id],
    NULL,
    NULL,
    &status
);
CHECK(status);
```

Шаг 4: Создание очереди команд

- `clCreateCommandQueue` используется для создания очереди команд, которая связана с выбранным устройством. Очередь команд используется для управления командами, которые отправляются на устройство для выполнения.

```
// Create a command queue using clCreateCommandQueue(),  
// and associate it with the device you want to execute on  
cmdQueue = clCreateCommandQueue(  
    context,  
    devices[device_id],  
    0,  
    &status  
);  
CHECK(status);
```


Шаг 5: Создание буферов устройств и копирование данных в них

- `clCreateBuffer` используется для создания буферов, которые будут использоваться для хранения входных и выходных данных на устройстве.
- `clEnqueueWriteBuffer` используется для копирования данных из памяти хоста (в данном случае, это массивы `matrix1` и `matrix2`) в буферы устройства (`bufferMatrixIn1` и `bufferMatrixIn2`).

```
cl_mem bufferMatrixIn1, bufferMatrixIn2, bufferMatrixOut;  
  
bufferMatrixIn1 = clCreateBuffer(  
    context,  
    CL_MEM_READ_ONLY,  
    size * size * sizeof(cl_double),  
    NULL,  
    &status  
);  
CHECK(status);
```

```
status = clEnqueueWriteBuffer(  
    cmdQueue,  
    bufferMatrixIn1,  
    CL_FALSE,  
    0,  
    size * size * sizeof(cl_double),  
    matrix1,  
    0,  
    NULL,  
    NULL  
);  
CHECK(status);
```

Шаг 6: Создание и компиляция OpenCL программы

```
const cl_program program = clCreateProgramWithSource(
    context,
    1,
    (const char **) &programBuffer,
    &programSize,
    &status
);
CHECK(status);

free(programBuffer);

// Build (compile) the program for the devices
const char options[] = "-cl-std=CL1.2";
status |= clBuildProgram(
    program,
    1,
    &devices[device_id],
    options,
    NULL,
    NULL
);
CHECK(status);
```

- Сначала каким-либо способом считывается файл с исходным кодом гостевой программы OpenCL (можно захардкодить).
- Затем создается программа OpenCL с помощью `clCreateProgramWithSource`, используя считанный исходный код.
- После этого программа компилируется для устройств с помощью `clBuildProgram`.

Шаг 7: создание ядра

- Ядро создается с помощью `clCreateKernel`. Ядро - это функция, написанная на языке OpenCL, которая выполняется на устройстве (в нашем случае, на GPU).

```
const cl_kernel mulKernel = clCreateKernel(  
    program,  
    "mul_kernel",  
    &status  
);  
CHECK(status);
```

Шаг 8: Установка аргументов ядра

- Аргументы ядра устанавливаются с помощью `clSetKernelArg`. В данном случае аргументами являются буферы устройства, которые были созданы ранее, и размер матрицы.

```
// Associate the input and output buffers with the kernel
status = clSetKernelArg(
    mulKernel,
    0,
    sizeof(cl_mem),
    &bufferMatrixIn1
);
CHECK(status);
```

```
status |= clSetKernelArg(
    mulKernel,
    2,
    sizeof(cl_mem),
    &bufferMatrixOut
);
CHECK(status);
```

Шаг 9: Настройка структуры рабочих элементов

- Здесь определяется глобальный размер рабочих элементов, который представляет собой общее количество рабочих элементов, которые будут выполнять ядро. В данном случае глобальный размер равен квадрату размера матрицы, так как каждый рабочий элемент будет вычислять один элемент результирующей матрицы.

```
size_t globalWorkSize[1];  
globalWorkSize[0] = size * size;
```

Шаг 9: Настройка структуры рабочих элементов

- Здесь определяется локальный размер рабочей группы, который представляет собой количество рабочих элементов в одной группе. Локальный размер может быть выбран разработчиком для оптимизации производительности, и он должен быть меньше или равен максимальному размеру рабочей группы, поддерживаемому устройством.

```
size_t localWorkSize[1];  
localWorkSize[0] = localSize;
```

Шаг 9: Настройка структуры рабочих элементов

- Команда добавляется в очередь команд, которая связана с определенным устройством.
- Ядро, которое нужно выполнить, передается в функцию вместе с его идентификатором.
- Размеры рабочих элементов и рабочих групп определяют, как будет распределено выполнение ядра. Глобальный размер рабочих элементов (`global_work_size`) указывает общее количество рабочих элементов, которые будут выполнять ядро. Локальный размер рабочей группы (`local_work_size`) определяет количество рабочих элементов в одной рабочей группе.
- Рабочие группы могут выполняться в любом порядке и могут быть назначены на любое ядро устройства. Это позволяет OpenCL оптимально распределять работу между ядрами устройства для повышения производительности.

```
status |= clEnqueueNDRangeKernel(  
    cmdQueue,  
    mulKernel,  
    1,  
    NULL,  
    globalWorkSize,  
    localWorkSize,  
    0,  
    NULL,  
    &mulDone  
);  
  
if (status != CL_SUCCESS) {  
    clWaitForEvents(1, &mulDone);  
    CHECK(status);  
}
```

Шаг 9: Настройка структуры рабочих элементов

- Эти функции используются для синхронизации выполнения ядра и чтения результатов из буфера на устройстве. `clWaitForEvents` блокирует выполнение до тех пор, пока не будет выполнено указанное событие (в данном случае `mulDone`), а `clEnqueueReadBuffer` асинхронно считывает данные из буфера на устройстве в буфер на хосте.

```
if (status != CL_SUCCESS) {  
    clWaitForEvents(1, &mulDone);  
    CHECK(status);  
}  
  
clEnqueueReadBuffer(  
    cmdQueue,  
    bufferMatrixOut,  
    CL_TRUE,  
    0,  
    size * size * sizeof(cl_double),  
    result_pl,  
    1,  
    &mulDone,  
    NULL  
);  
CHECK(status);
```


Шаг 10: Освобождение ресурсов OpenCL

- В конце все ресурсы OpenCL освобождаются с помощью функций `clRelease*`.

```
// Free OpenCL resources
clReleaseKernel(mulKernel);
clReleaseProgram(program);
clReleaseCommandQueue(cmdQueue);
clReleaseMemObject(bufferMatrixIn1);
clReleaseMemObject(bufferMatrixIn2);
clReleaseMemObject(bufferMatrixOut);
clReleaseContext(context);
```

size: 500, localSize: 50

```
Sequential execution 0.242136 (sec)
Parallel execution (local work group size: 50): 0.598501 (sec)
Kernel runtime: 0.020860 (sec).
Overhead 0.577641 (sec):
    Init time: 0.522043 (sec),
    Copy time: 0.000839 (sec),
    Compilation time: 0.054759 (sec)
```

size: 800, localSize: 50

```
Sequential execution 1.138443 (sec)
Parallel execution (local work group size: 50): 0.633299 (sec)
Kernel runtime: 0.108926 (sec).
Overhead 0.524373 (sec):
    Init time: 0.467660 (sec),
    Copy time: 0.000858 (sec),
    Compilation time: 0.055855 (sec)
```

size: 2300, localSize: 50

```
Sequential execution 110.619907 (sec)
```

```
Parallel execution (local work group size: 50): 4.185310 (sec)
```

```
Kernel runtime: 3.580529 (sec).
```

```
Overhead 0.604781 (sec):
```

```
    Init time: 0.538484 (sec),
```

```
    Copy time: 0.000549 (sec),
```

```
    Compilation time: 0.065748 (sec)
```

Спасибо за внимание

- Ссылка на репозиторий:
https://github.com/stdStudent/ocl_example
- Мануал OpenCL:
<https://github.com/KhronosGroup/OpenCL-Guide>

