Программирование на видеокартах

Начало

Как это всё компилировать

Хост-код

.h файлы для C/C++ есть на официальном сайте, правда, там есть задепрекейченные полезные функции. Придётся задефайнить версию. Достаточно хедера `<CL/cl.h>`. Можно — `<CL/opencl.h>`

Чтобы слиноваться, надо где-то взять либы. Под линуксом — некий магический package. Под виндоусом — откуда-то взять. На официальном сайте Кроноса есть. Обратите внимание на битность!

Девайс-код

Само разберётся. Обычно.

Запуск

В пределах конспекта код запускается с помощью cmake:

```
cmake_minimum_required(VERSION 3.28)
set(CMAKE_CXX_STANDARD 20)

project(Main)
add_executable(Main main.cpp)
target_compile_features(Main PRIVATE cxx_auto_type)

find_package(OpenCL REQUIRED)
target_link_libraries(Main OpenCL::OpenCL)
```

Как водится в плюсах, придётся написать некоторое количество функций и (тьфу-тьфу) макросов, чтобы было удобно. Полный перечень того, что использую я, можно прочитать в файле “sources/aux/gpu-prelude”.

Пытаемся запустить:

```
std::vector v({1, 2, 3});
println(v);
```

```
Compilation error
In file included from /usr/include/CL/cl.h:20,
[ 50%] Building CXX object CMakeFiles/Main.dir/main.cpp.o
[100%] Linking CXX executable Main
[100%] Built target Main
                 from /home/ldemetrios/Workspace/Conspects4sem/__tmp__/main.cpp:5:
/usr/include/CL/cl_version.h:22:104: note: â€˜#pragma message: cl_version.h: CL_TARGET_OPENCL_VERSION is not defined. Defaulting to 300 (OpenCL 3.0)â€™
   22 | #pragma message("cl_version.h: CL_TARGET_OPENCL_VERSION is not defined. Defaulting to 300 (OpenCL 3.0)")
      |                                                                                                        ^
```

Работает.

API

Большинство функций возвращают код ошибки (как обычно, 0 — успешно, не 0 — не успешно). Надо проверять! Во всяком случае, в дебаге, в учебных целях. Написать макрос? Какой ужас.

Это за исключением функций типа `create`, которые возвращают то, что они `create`. Тогда код ошибки, если нужен, по ссылке, передаваемой в аргумент… Такой типичный C.

Получение девайсов

clGetPlatformIDs

— API для получения списка доступных платформ. Принимает… Указатель, размер, и указатель на размер… Буффер, размер буффера, и то, куда записывать, сколько.

Понятное дело, память она не выделяет. Так как надо, чтобы освобождал тот, кто выделил. Такой типичный C…

Так, а какого размера выделять буффер? Для этого есть специальный вариант вызова: `(nullptr, 0, &x)` — тогда в x нам запишут то, сколько на самом деле вариантов. Идейно. Такой принцип применяется здесь во всех вызовах.

Правда, в отличие от типичного C, если буфера мало, то это не ошибка “буфера мало”, а нам запишут, сколько есть места.

Можно сказать, “возвращает” эта функция список id платформ, где можно запустить opencl.

```
uint32_t s;
clGetPlatformIDs(0, nullptr, &s);
std::vector<cl_platform_id> platforms(s);
clGetPlatformIDs(s, platforms.data(), &s);
println(platforms);
```

```
Compilation error
In file included from /usr/include/CL/cl.h:20,
[ 50%] Building CXX object CMakeFiles/Main.dir/main.cpp.o
[100%] Linking CXX executable Main
[100%] Built target Main
                 from /home/ldemetrios/Workspace/Conspects4sem/__tmp__/main.cpp:5:
/usr/include/CL/cl_version.h:22:104: note: â€˜#pragma message: cl_version.h: CL_TARGET_OPENCL_VERSION is not defined. Defaulting to 300 (OpenCL 3.0)â€™
   22 | #pragma message("cl_version.h: CL_TARGET_OPENCL_VERSION is not defined. Defaulting to 300 (OpenCL 3.0)")
      |                                                                                                        ^
```

Результат не вполне содержательный — `cl_platform_id` это просто указатель на не очень понятно что. Но нам это и не надо — это всего лишь идентификатор, который умеют понимать другие функции апи.

clGetPlatformInfo

— Получаем информацию о конкретной платформе. Сюда передаётся `platformID`, полученный на предыдущем этапе, и константа, обозначающая, какую информацию мы хотим. Остальное — как раньше.

```
for (auto platform : platforms) {
    println(platform);
    size_t pl_name_length;
    clGetPlatformInfo(platform, CL_PLATFORM_NAME, 0, nullptr, &pl_name_length);
    std::vector<char> pl_name(pl_name_length);
    clGetPlatformInfo(platform, CL_PLATFORM_NAME, pl_name_length, pl_name.data(),
                      &pl_name_length);
    std::string str(pl_name.begin(), pl_name.end());
    println(str);
}
```

```
Compilation error
In file included from /usr/include/CL/cl.h:20,
[ 50%] Building CXX object CMakeFiles/Main.dir/main.cpp.o
[100%] Linking CXX executable Main
[100%] Built target Main
                 from /home/ldemetrios/Workspace/Conspects4sem/__tmp__/main.cpp:5:
/usr/include/CL/cl_version.h:22:104: note: â€˜#pragma message: cl_version.h: CL_TARGET_OPENCL_VERSION is not defined. Defaulting to 300 (OpenCL 3.0)â€™
   22 | #pragma message("cl_version.h: CL_TARGET_OPENCL_VERSION is not defined. Defaulting to 300 (OpenCL 3.0)")
      |                                                                                                        ^
```

Здесь можно запросить разную информацию. Но сначала давайте сначала сделаем какую-нибудь обёртку для всех этих вызовов. Можно было бы сделать это по-честному template функцией, но мы хотим без лишней возни удобные сообщения об ошибках…

```
#define request_error_message(func, ...) "Request " #func "(" #__VA_ARGS__ __VA_OPT__(", ") "...) terminated with non-zero exit code "

#define request(SizeT, T, func, ...) cl_call<SizeT, T>(&func, request_error_message(func __VA_OPT__(,) __VA_ARGS__ ) __VA_OPT__(,) __VA_ARGS__)
```

Выколите мне глаза. И таких ещё парочку. Смотрите сами знаете где.

Зато теперь относительно нормально вызываем:

```
std::vector<cl_platform_id> platforms = request(uint32_t, cl_platform_id, clGetPlatformIDs);
println(platforms);
println();

for (auto platform : platforms) {
    println(platform);
    auto profile = request_str(clGetPlatformInfo, platform, CL_PLATFORM_PROFILE);
    auto version = request_str(clGetPlatformInfo, platform, CL_PLATFORM_VERSION);
    auto name = request_str(clGetPlatformInfo, platform, CL_PLATFORM_NAME);
    auto vendor = request_str(clGetPlatformInfo, platform, CL_PLATFORM_VENDOR);
    auto extensions = request_str(clGetPlatformInfo, platform, CL_PLATFORM_EXTENSIONS);
    auto host_timer_resolution = request(size_t, cl_ulong, clGetPlatformInfo, platform,
                                         CL_PLATFORM_HOST_TIMER_RESOLUTION);
    println("Profile:               ", profile);
    println("Version:               ", version);
    println("Name:                  ", name);
    println("Vendor:                ", vendor);
    println("Extensions:            ", extensions);
    println("Host Timer Resolution: ", host_timer_resolution);
}
```

```
Compilation error
In file included from /usr/include/CL/cl.h:20,
[ 50%] Building CXX object CMakeFiles/Main.dir/main.cpp.o
[100%] Linking CXX executable Main
[100%] Built target Main
                 from /home/ldemetrios/Workspace/Conspects4sem/__tmp__/main.cpp:5:
/usr/include/CL/cl_version.h:22:104: note: â€˜#pragma message: cl_version.h: CL_TARGET_OPENCL_VERSION is not defined. Defaulting to 300 (OpenCL 3.0)â€™
   22 | #pragma message("cl_version.h: CL_TARGET_OPENCL_VERSION is not defined. Defaulting to 300 (OpenCL 3.0)")
      |                                                                                                        ^
```

clGetDeviceIDs

Наконец, мы хотим получить от платформы список доступных девайсов. Эта функция принимает id платформы и тип девайса, который мы хотим, на выбор:

CL_DEVICE_TYPE_CPU	An OpenCL device that is the host processor. The host processor runs the OpenCL implementations and is a single or multi-core CPU.
CL_DEVICE_TYPE_GPU	An OpenCL device that is a GPU. By this we mean that the device can also be used to accelerate a 3D API such as OpenGL or DirectX.
CL_DEVICE_TYPE_ACCELERATOR	Dedicated OpenCL accelerators (for example the IBM CELL Blade). These devices communicate with the host processor using a peripheral interconnect such as PCIe.
CL_DEVICE_TYPE_DEFAULT	The default OpenCL device in the system.
CL_DEVICE_TYPE_ALL	All OpenCL devices available in the system.

– Прямиком из документации

CL_DEVICE_TYPE_CPU	Устройство OpenCL, главный процессор. Хост-процессор запускает реализации OpenCL и представляет собой одно– или многоядерный CPU.
CL_DEVICE_TYPE_GPU	Устройство OpenCL, графический процессор. Под этим мы подразумеваем, что устройство также можно использовать для ускорения 3D API, такого как OpenGL или DirectX.
CL_DEVICE_TYPE_ACCELERATOR	Специальные ускорители OpenCL (например, IBM CELL Blade). Эти устройства взаимодействуют с главным процессором с помощью периферийного соединения, такого как PCIe.
CL_DEVICE_TYPE_DEFAULT	Устройство OpenCL по умолчанию в системе.
CL_DEVICE_TYPE_ALL	Все устройства OpenCL, доступные в системе.

– Мне лень переводить.

```
for (auto platform : platforms) {
    println("For platform ", platform);
    println("CPU:         ", request(cl_uint, cl_device_id, clGetDeviceIDs, platform, CL_DEVICE_TYPE_CPU));
    println("GPU:         ", request(cl_uint, cl_device_id, clGetDeviceIDs, platform, CL_DEVICE_TYPE_GPU));
    println("ACCELERATOR: ", request(cl_uint, cl_device_id, clGetDeviceIDs, platform, CL_DEVICE_TYPE_ACCELERATOR));
    println("DEFAULT:     ", request(cl_uint, cl_device_id, clGetDeviceIDs, platform, CL_DEVICE_TYPE_DEFAULT));
    println("ALL:         ", request(cl_uint, cl_device_id, clGetDeviceIDs, platform, CL_DEVICE_TYPE_ALL));
}
```

```
Compilation error
In file included from /usr/include/CL/cl.h:20,
[ 50%] Building CXX object CMakeFiles/Main.dir/main.cpp.o
[100%] Linking CXX executable Main
[100%] Built target Main
                 from /home/ldemetrios/Workspace/Conspects4sem/__tmp__/main.cpp:5:
/usr/include/CL/cl_version.h:22:104: note: â€˜#pragma message: cl_version.h: CL_TARGET_OPENCL_VERSION is not defined. Defaulting to 300 (OpenCL 3.0)â€™
   22 | #pragma message("cl_version.h: CL_TARGET_OPENCL_VERSION is not defined. Defaulting to 300 (OpenCL 3.0)")
      |                                                                                                        ^
```

Здесь внезапно обнаруживаем, что если девайсов не найдено, то первый вызов функции не “вернёт” 0, а выдаст код ошибки −1. Учитываем это.

Ну круто, у нас одна платформа с одним девайсом. Зато выбирать не надо!

Делаем что-то полезное

Допустим, мы выбрали девайс, который нам нравится, с которым мы хотим работать.

```
std::vector<cl_device_id> devices = request(
    cl_uint, cl_device_id, clGetDeviceIDs, platforms[0], CL_DEVICE_TYPE_ALL
);
```

```
Compilation error
In file included from /usr/include/CL/cl.h:20,
[ 50%] Building CXX object CMakeFiles/Main.dir/main.cpp.o
[100%] Linking CXX executable Main
[100%] Built target Main
                 from /home/ldemetrios/Workspace/Conspects4sem/__tmp__/main.cpp:5:
/usr/include/CL/cl_version.h:22:104: note: â€˜#pragma message: cl_version.h: CL_TARGET_OPENCL_VERSION is not defined. Defaulting to 300 (OpenCL 3.0)â€™
   22 | #pragma message("cl_version.h: CL_TARGET_OPENCL_VERSION is not defined. Defaulting to 300 (OpenCL 3.0)")
      |                                                                                                        ^
```

… или несколько…

Теперь нам нужно создать контекст.

clCreateContext

Контекст — это своего рода `globalThis` от мира OpenCL. Он инкапсулирует в себе всё, что мы хотим из хост-кода делать с девайс-кодом.

Функция принимает кучу всего. В частности, умеет принимать несколько девайсов, если они принадлежат одной платформе. Ну, пока обойдёмся. В качестве `user_data` передаём nullptr. `properties` — тоже странная штука, пока обойдёмся без неё.

```
cl_int errcode;
cl_context context = clCreateContext(
    nullptr, devices.size(), devices.data(), nullptr, nullptr, &errcode
);
println("Error code: ", errcode);
println("Context: ", context);
```

```
Compilation error
In file included from /usr/include/CL/cl.h:20,
[ 50%] Building CXX object CMakeFiles/Main.dir/main.cpp.o
[100%] Linking CXX executable Main
[100%] Built target Main
                 from /home/ldemetrios/Workspace/Conspects4sem/__tmp__/main.cpp:5:
/usr/include/CL/cl_version.h:22:104: note: â€˜#pragma message: cl_version.h: CL_TARGET_OPENCL_VERSION is not defined. Defaulting to 300 (OpenCL 3.0)â€™
   22 | #pragma message("cl_version.h: CL_TARGET_OPENCL_VERSION is not defined. Defaulting to 300 (OpenCL 3.0)")
      |                                                                                                        ^
```

Ещё один несодержательный указатель. Ах да, и нам нужен новый макрос.

```
auto context = create(
    cl_context, clCreateContext, nullptr, devices.size(), devices.data(), nullptr, nullptr
);
println(context);
```

```
Compilation error
In file included from /usr/include/CL/cl.h:20,
[ 50%] Building CXX object CMakeFiles/Main.dir/main.cpp.o
[100%] Linking CXX executable Main
[100%] Built target Main
                 from /home/ldemetrios/Workspace/Conspects4sem/__tmp__/main.cpp:5:
/usr/include/CL/cl_version.h:22:104: note: â€˜#pragma message: cl_version.h: CL_TARGET_OPENCL_VERSION is not defined. Defaulting to 300 (OpenCL 3.0)â€™
   22 | #pragma message("cl_version.h: CL_TARGET_OPENCL_VERSION is not defined. Defaulting to 300 (OpenCL 3.0)")
      |                                                                                                        ^
```

Теперь нужно в этом контексте создать тот код, который мы будем запускать в этом контексте:

clCreateProgramWithSource

“Принимает” массив указателей на source.

Обычно мы хотим не извращаться с C-style строчками, а иметь отдельный файл с исходным кодом (обычно расширение `.cl`). То есть, надо открыть файл, прочитать и скормить (компиляция девайс-кода будет уже в рантайме). Предлагается использовать обычные С’шные функции fread и прочее. Открыть в `binary`, `seek` до конца и так далее.

Эта функция не делает ничего особенного. Если мы тут зафейлились, мы где-то конкретно налажали — передали кривой буфер или ещё что-нибудь. Ошибки внутри девайс-кода будут обнаруживаться уже потом.

clBuildProgram

Вот это уже серьёзно: компиляция нашего исходника. Передаём сюда `id` нашего `program`’а, который мы по’create’или, и девайс, под который надо скомпилиться. Или передать null, и тогда будет скомпилированно под все девайсы, привязанные к контексту.

Компиляция может занимать продолжительное время! Имеет смысл локально закэшировать. Но так как результат сильно зависит от драйверов, девайсов, версий, погоды, фазы луны; распространять прекомпилированную версию не имеет смысла.

Если в девайс-коде есть какая-нибудь синтаксическая (или ещё что-нибудь) ошибка, то она вылезет здесь. Обязательно проверять результат здесь! Можно сделать `clGetBuildInfo`, чтобы получить билд-лог (своего рода compilation error), ему нужно давать честный `id` девайса.

build-options — не должен быть null! Некоторые платформы могут его не пережить. Передайте пустую строчку. Или, лучше, используйте по назначению! Например, можно с помощью `-D` передавать дефайнами переменные.

Наконец, девайс-код

```
kernel void add(global const int *a, global const int *b, global int *c) {
  *c = *a + *b;
}
```

Здесь немного нового по сравнению с C.

- `kernel` означает, что это точка входа в программу. Их может быть несколько — это не совсем то же, что `main`.
- `global` означает (что означает?)

Давайте делать.

```
const char* source = "kernel void add(global const int *a, "
                                            "global const int *b, global int *c) {\n"
    "    *c = *a + *b;\n"
    "}\n";
std::vector<const char*> source_arr{source};

std::vector<size_t> source_lengths(source_arr.size());
std::transform(
    source_arr.begin(), source_arr.end(), source_lengths.begin(),
    [](const char* s) -> size_t { return std::string(s).length(); });

cl_program program = create(cl_program, clCreateProgramWithSource, context,
                            source_arr.size(), source_arr.data(), source_lengths.data());

println(program);

auto err = clBuildProgram(program, devices.size(), devices.data(), "", nullptr, nullptr);
if(err != 0) {
    println("clBuildProgram returned with non-zero code: ", err);
    for(auto device : devices) {
        println();
        println("Build info for device ", device, ":");
        println(request_str(clGetProgramBuildInfo, program, device, CL_PROGRAM_BUILD_LOG));
    }
}
```

```
Compilation error
In file included from /usr/include/CL/cl.h:20,
[ 50%] Building CXX object CMakeFiles/Main.dir/main.cpp.o
[100%] Linking CXX executable Main
[100%] Built target Main
                 from /home/ldemetrios/Workspace/Conspects4sem/__tmp__/main.cpp:5:
/usr/include/CL/cl_version.h:22:104: note: â€˜#pragma message: cl_version.h: CL_TARGET_OPENCL_VERSION is not defined. Defaulting to 300 (OpenCL 3.0)â€™
   22 | #pragma message("cl_version.h: CL_TARGET_OPENCL_VERSION is not defined. Defaulting to 300 (OpenCL 3.0)")
      |                                                                                                        ^
```

… и ещё один указатель куда-то…

Зато, если у нас не удался `build`, мы получаем человекочитаемое сообщение об ошибке!

Например, попытаемся вставить пробел в середину слова `void`:

```
const char* source = "kernel vo id add(global const int *a, "
 "global const int *b, global int *c) {\n"
 " *c = *a + *b;\n"
 "}\n";
std::vector<const char*> source_arr{source};

std::vector<size_t> source_lengths(source_arr.size());
std::transform(
 source_arr.begin(), source_arr.end(), source_lengths.begin(),
 [](const char* s) -> size_t { return std::string(s).length(); });

cl_program program = create(cl_program, clCreateProgramWithSource, context,
 source_arr.size(), source_arr.data(), source_lengths.data());

auto err = clBuildProgram(program, devices.size(), devices.data(), "", nullptr,
nullptr);
if(err != 0) {
 println("clBuildProgram returned with non-zero code: ", err);
 for(auto device : devices) {
 println();
 println("Build info for device ", device, ":");
 println(request_str(clGetProgramBuildInfo, program, device,
CL_PROGRAM_BUILD_LOG));
 }
}
```

```
Compilation error
In file included from /usr/include/CL/cl.h:20,
[ 50%] Building CXX object CMakeFiles/Main.dir/main.cpp.o
[100%] Linking CXX executable Main
[100%] Built target Main
                 from /home/ldemetrios/Workspace/Conspects4sem/__tmp__/main.cpp:5:
/usr/include/CL/cl_version.h:22:104: note: â€˜#pragma message: cl_version.h: CL_TARGET_OPENCL_VERSION is not defined. Defaulting to 300 (OpenCL 3.0)â€™
   22 | #pragma message("cl_version.h: CL_TARGET_OPENCL_VERSION is not defined. Defaulting to 300 (OpenCL 3.0)")
      |                                                                                                        ^
```

clCreateKernel

— создаёт идентификатор, через который мы сможем вызывать `kernel`. Передаём туда имя.

```
cl_kernel add = create(cl_kernel, clCreateKernel, program, "add");
```

```
Compilation error
In file included from /usr/include/CL/cl.h:20,
[ 50%] Building CXX object CMakeFiles/Main.dir/main.cpp.o
[100%] Linking CXX executable Main
[100%] Built target Main
                 from /home/ldemetrios/Workspace/Conspects4sem/__tmp__/main.cpp:5:
/usr/include/CL/cl_version.h:22:104: note: â€˜#pragma message: cl_version.h: CL_TARGET_OPENCL_VERSION is not defined. Defaulting to 300 (OpenCL 3.0)â€™
   22 | #pragma message("cl_version.h: CL_TARGET_OPENCL_VERSION is not defined. Defaulting to 300 (OpenCL 3.0)")
      |                                                                                                        ^
```

Теперь мы хотим передать ему аргументы и запустить его! Проблемы? Э-э-э……… Указатели на память, что вы думаете, сработает? Ха. Где память? На девайсе. Мы её выделили? Нет. Очень жаль. Давайте выделять…

Here we go again

clCreateBuffer

Выделяем память на девайсе. Нам надо только передать размер буфера, и то, как мы хотим к нему обращаться из кернела (read only, write only, read-write). В данном случае, мы хотим три буфера: под `a` и под `b` — read-only, под `c` — write-only. Ну, мы можем оба подписать read-write, но лучше так — так что-нибудь оптимизировать (не надо про протокол когерентности, кэши, всё остальное думать).

Понятно, что доступ распространяется только на кернел, с хоста-то мы и так записать/прочитать сможем.

`clCreateBuffer` возвращает `cl_mem`, это своего рода указатель, но не указатель. Это хендл, арифметику с ним делать нельзя.

А мы написали `int`… а что это? Понятное дело, что `int` на девайсе и `int` на хосте могут отличаться. Ну так для этого может быть `cl_int`. На самом деле, они даже фиксированы, не зависят от девайса. Так что это всегда 32 бита. Лучше, чем в C! А вот `size_t` девайса мы не знаем. Точнее, можем спросить, но это будет уже в рантайме и у конкретного девайса. Так что кернелам просто запрещено принимать `size_t`.

```
cl_mem aBuffer = create(cl_mem, clCreateBuffer, context, CL_MEM_READ_ONLY, 4, nullptr);
cl_mem bBuffer = create(cl_mem, clCreateBuffer, context, CL_MEM_READ_ONLY, 4, nullptr);
cl_mem cBuffer = create(cl_mem, clCreateBuffer, context, CL_MEM_WRITE_ONLY, 4, nullptr);
```

```
Compilation error
In file included from /usr/include/CL/cl.h:20,
[ 50%] Building CXX object CMakeFiles/Main.dir/main.cpp.o
[100%] Linking CXX executable Main
[100%] Built target Main
                 from /home/ldemetrios/Workspace/Conspects4sem/__tmp__/main.cpp:5:
/usr/include/CL/cl_version.h:22:104: note: â€˜#pragma message: cl_version.h: CL_TARGET_OPENCL_VERSION is not defined. Defaulting to 300 (OpenCL 3.0)â€™
   22 | #pragma message("cl_version.h: CL_TARGET_OPENCL_VERSION is not defined. Defaulting to 300 (OpenCL 3.0)")
      |                                                                                                        ^
```

clSetKernelArg

Наконец, можем накормить кернел аргументами. Указываем сюда идентификатор кернела, номер аргумента, значение аргумента (оно — как адрес + количество байтов). Так как мы передаём “указатели”, в качестве количества байтов у нас `size_of(cl_mem)`

```
err = clSetKernelArg(add, 0, sizeof(cl_mem), &aBuffer);
if (err != 0) throw exec_error("Can't set 0th kernel arg");
err = clSetKernelArg(add, 1, sizeof(cl_mem), &bBuffer);
if (err != 0) throw exec_error("Can't set 1st kernel arg");
err = clSetKernelArg(add, 2, sizeof(cl_mem), &cBuffer);
if (err != 0) throw exec_error("Can't set 2nd kernel arg");
```

```
Compilation error
In file included from /usr/include/CL/cl.h:20,
[ 50%] Building CXX object CMakeFiles/Main.dir/main.cpp.o
[100%] Linking CXX executable Main
[100%] Built target Main
                 from /home/ldemetrios/Workspace/Conspects4sem/__tmp__/main.cpp:5:
/usr/include/CL/cl_version.h:22:104: note: â€˜#pragma message: cl_version.h: CL_TARGET_OPENCL_VERSION is not defined. Defaulting to 300 (OpenCL 3.0)â€™
   22 | #pragma message("cl_version.h: CL_TARGET_OPENCL_VERSION is not defined. Defaulting to 300 (OpenCL 3.0)")
      |                                                                                                        ^
```

clEnqueueWriteBuffer

В наших буферах лежит какой-то мусор, надо его наполнить. Чувствуется в названии подвох.

clCreateCommandQueue

— очередь команд, чего мы хотим. Принимает контекст и девайс. А ещё принимает *флажки*. Один из них полезный — profiling info (или как-то так), причём почти ничего не стоит. Рассказывает, сколько времени уходит на процессы. Второй — `out of order executionary mode`. Не влезай, убьёт.

```
auto command_queue = create(
    cl_command_queue, clCreateCommandQueue, context, devices[0], CL_QUEUE_PROFILING_ENABLE
);
```

```
Compilation error
In file included from /usr/include/CL/cl.h:20,
[ 50%] Building CXX object CMakeFiles/Main.dir/main.cpp.o
[100%] Linking CXX executable Main
[100%] Built target Main
                 from /home/ldemetrios/Workspace/Conspects4sem/__tmp__/main.cpp:5:
/usr/include/CL/cl_version.h:22:104: note: â€˜#pragma message: cl_version.h: CL_TARGET_OPENCL_VERSION is not defined. Defaulting to 300 (OpenCL 3.0)â€™
   22 | #pragma message("cl_version.h: CL_TARGET_OPENCL_VERSION is not defined. Defaulting to 300 (OpenCL 3.0)")
      |                                                                                                        ^
/home/ldemetrios/Workspace/Conspects4sem/__tmp__/main.cpp: In function â€˜int main()â€™:
/home/ldemetrios/Workspace/Conspects4sem/__tmp__/main.cpp:139:23: warning: â€˜_cl_command_queue* clCreateCommandQueue(cl_context, cl_device_id, cl_command_queue_properties, cl_int*)â€™ is deprecated [-Wdeprecated-declarations]
  139 |         cl_command_queue, clCreateCommandQueue, context, devices[0], CL_QUEUE_PROFILING_ENABLE
      |                           ^~~~~~~~~~~~~~~~~~~~
/home/ldemetrios/Workspace/Conspects4sem/__tmp__/main.cpp:87:44: note: in definition of macro â€˜createâ€™
   87 | #define create(T, func, ...) cl_create<T>(&func, generate_error_message(func __VA_OPT__(,) __VA_ARGS__ ) __VA_OPT__(,) __VA_ARGS__)
      |                                            ^~~~
/usr/include/CL/cl.h:1920:1: note: declared here
 1920 | clCreateCommandQueue(cl_context                     context,
      | ^~~~~~~~~~~~~~~~~~~~
```

clEnqueueWriteBuffer

, да. Принимает `cl_mem`, указатель наш (откуда брать данные), флажок `blocking write`. Про блокирующее чтение и запись. Мы ставим *задание на постановку в очередь*. Если мы не поставим флажок, то оно вернётся мгновенно! И ничего не дождётся. Имеет смысл делать передачу данных **на** девайс не блокирующей, а **с** девайса — блокирующей. Очередь ленивая, не будет ничего исполнять, пока мы не пнём её. Ну, или можно пнуть через “подождать выполнения всех функций”. Или спросить “~~а не в омах ли измеряется сопротивление~~ а не закончилось ли исполнение”.

- Спойлер: В конце захотим `clEnqueueReadBuffer`.

```
cl_int a = 2;
cl_int b = 3;

call(clEnqueueWriteBuffer, command_queue, aBuffer, false, 0, 4, &a, 0, nullptr, nullptr);
call(clEnqueueWriteBuffer, command_queue, bBuffer, false, 0, 4, &b, 0, nullptr, nullptr);
```

```
Compilation error
In file included from /usr/include/CL/cl.h:20,
[ 50%] Building CXX object CMakeFiles/Main.dir/main.cpp.o
[100%] Linking CXX executable Main
[100%] Built target Main
                 from /home/ldemetrios/Workspace/Conspects4sem/__tmp__/main.cpp:5:
/usr/include/CL/cl_version.h:22:104: note: â€˜#pragma message: cl_version.h: CL_TARGET_OPENCL_VERSION is not defined. Defaulting to 300 (OpenCL 3.0)â€™
   22 | #pragma message("cl_version.h: CL_TARGET_OPENCL_VERSION is not defined. Defaulting to 300 (OpenCL 3.0)")
      |                                                                                                        ^
/home/ldemetrios/Workspace/Conspects4sem/__tmp__/main.cpp: In function â€˜int main()â€™:
/home/ldemetrios/Workspace/Conspects4sem/__tmp__/main.cpp:139:23: warning: â€˜_cl_command_queue* clCreateCommandQueue(cl_context, cl_device_id, cl_command_queue_properties, cl_int*)â€™ is deprecated [-Wdeprecated-declarations]
  139 |         cl_command_queue, clCreateCommandQueue, context, devices[0], CL_QUEUE_PROFILING_ENABLE
      |                           ^~~~~~~~~~~~~~~~~~~~
/home/ldemetrios/Workspace/Conspects4sem/__tmp__/main.cpp:87:44: note: in definition of macro â€˜createâ€™
   87 | #define create(T, func, ...) cl_create<T>(&func, generate_error_message(func __VA_OPT__(,) __VA_ARGS__ ) __VA_OPT__(,) __VA_ARGS__)
      |                                            ^~~~
/usr/include/CL/cl.h:1920:1: note: declared here
 1920 | clCreateCommandQueue(cl_context                     context,
      | ^~~~~~~~~~~~~~~~~~~~
```

clEnqueueNDRangeKernel — ураа!

Запуск кернела. Передаём туда, очевидно, очередь и кернел. Принимает также dimensions (who? пока передаём 1); global work size, причём через указатель (пока тоже 1); local work size — смело кормим null’ами, так же поступаем с event’ами.

```
size_t global_work_size = 1;
call(clEnqueueNDRangeKernel, command_queue, add, 1,
        nullptr, &global_work_size, nullptr, 0, nullptr, nullptr);
```

```
Compilation error
In file included from /usr/include/CL/cl.h:20,
[ 50%] Building CXX object CMakeFiles/Main.dir/main.cpp.o
[100%] Linking CXX executable Main
[100%] Built target Main
                 from /home/ldemetrios/Workspace/Conspects4sem/__tmp__/main.cpp:5:
/usr/include/CL/cl_version.h:22:104: note: â€˜#pragma message: cl_version.h: CL_TARGET_OPENCL_VERSION is not defined. Defaulting to 300 (OpenCL 3.0)â€™
   22 | #pragma message("cl_version.h: CL_TARGET_OPENCL_VERSION is not defined. Defaulting to 300 (OpenCL 3.0)")
      |                                                                                                        ^
/home/ldemetrios/Workspace/Conspects4sem/__tmp__/main.cpp: In function â€˜int main()â€™:
/home/ldemetrios/Workspace/Conspects4sem/__tmp__/main.cpp:139:23: warning: â€˜_cl_command_queue* clCreateCommandQueue(cl_context, cl_device_id, cl_command_queue_properties, cl_int*)â€™ is deprecated [-Wdeprecated-declarations]
  139 |         cl_command_queue, clCreateCommandQueue, context, devices[0], CL_QUEUE_PROFILING_ENABLE
      |                           ^~~~~~~~~~~~~~~~~~~~
/home/ldemetrios/Workspace/Conspects4sem/__tmp__/main.cpp:87:44: note: in definition of macro â€˜createâ€™
   87 | #define create(T, func, ...) cl_create<T>(&func, generate_error_message(func __VA_OPT__(,) __VA_ARGS__ ) __VA_OPT__(,) __VA_ARGS__)
      |                                            ^~~~
/usr/include/CL/cl.h:1920:1: note: declared here
 1920 | clCreateCommandQueue(cl_context                     context,
      | ^~~~~~~~~~~~~~~~~~~~
```

clEnqueueReadBuffer

Наконец, читаем нашу замечательную сумму двух чисел. Будем на это, во всяком случае, надеяться.

- Замечание к окончанию: всё, что мы `create`, хорошо бы потом `release`. А то может быть грустно.
- А если что-нибудь криво работает на девайсе… Ну, упадёт видео-драйвер. Винда его обычно поднимает через пару секунд, остальные — … поэкспериментируйте!

> **TODO!**
> Полный код

> **TODO!**
> Переписать на нормальные функции

Наконец, занимаемся чем-то полезным.

Хотим посчитать сумму элементов массива.

- work_item — идейно тред. Но их много. Тысячи их.
- SINT: single instruction, multiple threads. Есть вычислительные блоки, а есть понятие “ядер” видеокарты. Если проводить аналогию с процессором — ядер не так много. Quda-псевдо-ядра это что-то в духе конвеером (умеет считать, но не имеет логики управления). Конвееры объединяются в ядра, как раз. И вот эти work-item’ы — запускаются на конвейерах. И они исполняют одну и ту же команду одновременно. Если у нас ситуация, что есть if, то сначала все исполняют одну ветку (часть тредов ждёт), потом наоборот. Хотя если все одновременно в одну ветку зашли, то ждать не будут. Особенно интересно про циклы будет, конечно…
- Соответственно, пачку тредов, исполняющихся одновременно, называем варпом. Их там 32 или 64, но это не очень важно. Где-то зашито, где-то можно выбрать. Это набор тредов, выполняющихся аппаратно синхронно. Хотя на новых видеокарточках, бывает, расходится, ну да неважно.
- Сэкономили на обращении к памяти. Запускаем на ядре несколько пачек тредов, и переключаемся между ними, когда текущая обращается к памяти. Поэтому принципиально кэш видеокарточкам не нужен, но кое-где его делают. Где-то для буферизации записи, где-то для экономии энергии.
- Так вот, `global_work_size` — это то, сколько тредов мы хотим запустить *всего*. Они, конечно, не обязательно будут запущены одновременно. `local_work_size` — гарантированно будет запущенно на одном аппаратном исполнителе. Зачем? У тредов есть оперативка девайса (global), и регистры (одного потока, их мало). А ещё! Есть локальная память (local). Она общая для тредов одного исполнителя.

На размер локальной группы есть ограничения, можно спросить у девайса, какой у него лимит. По стандарту нам гарантируется, что максимальный — … адын! Хотя на самом деле обычно 256, а DirectX и вовсе требует 1024. Ну и соответственно, если сделать некратно размеру варпа, то просто часть тредов будет неактивна.

И так, что мы теперь делаем, чтобы запустить наше `a+b`?

1. Буфера размера побольше.
2. Global work size размером в количество элементов.
3. Мы не используем никакого межтредового взаимодействия, так что `local_work_size` — `null`. То есть, драйвер, разбирайся сам.
4. Получить внутри кернела его номер: `get_global_id(0)` (возвращает size_t)
5. Наконец, поменять кернел:

```
size_t x = get_global_id(0);
c[x] = a[x] + b[x];
```

Правда, регистры здесь все 32-битные, а size_t не обязательно 32-битное. А мы хотим не выжирать много регистров (приватной памяти), а то часть ещё пойдёт в глобальную память… Так что будьте внимательны, может, лучше использовать иногда uint. Кстати, можно узнать, сколько у нас не влезло в регистры, и поправить код. А то это самая большая просадка по времени будет.

Аналогично с `get_global_id`, есть `get_local_id` — номер в группе, и `get_group_id` — номер группы.

А почему 0? А помните у нас был dimension — 1? Это нумерация тредов. Она может быть одномерной, двумерной, трёхмерной. Соответственно, тогда у треда будет номер по x, номер по y, номер по z. Просто для удобства нумерации, чтобы не надо было. Так вот 0 — это номер координаты.