Настройка проекта OpenCL

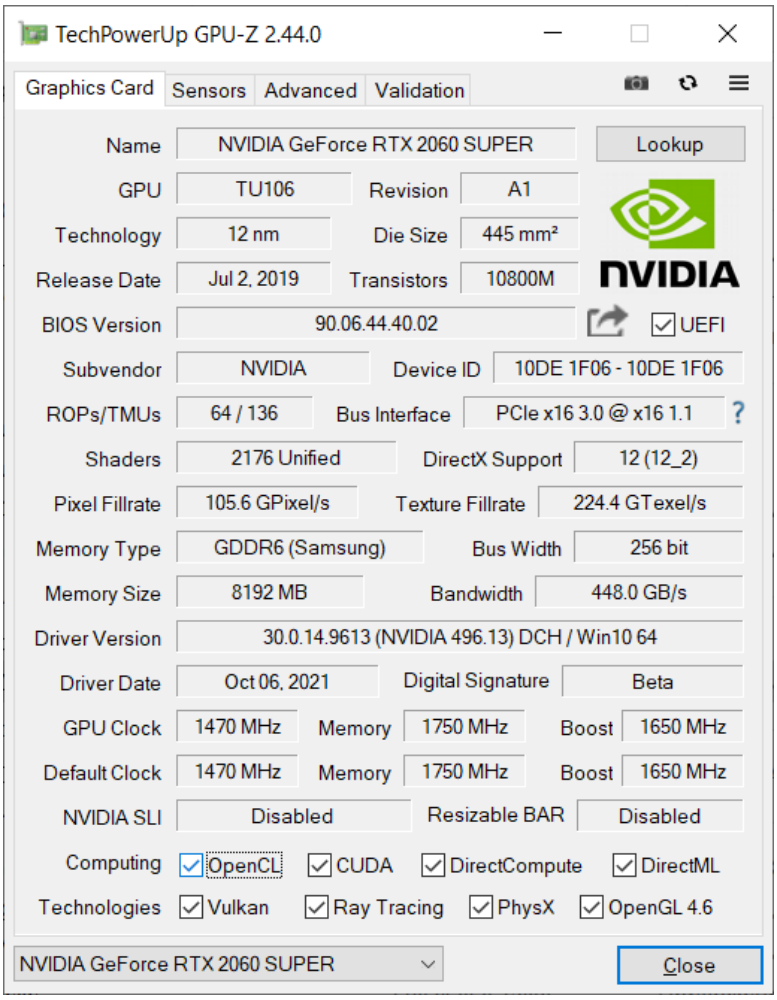
САМОСТОЯТЕЛЬНАЯ РАБОТА

http://www.kimrt.ru

Онлайн-курс

Суперкомпьютерные технологии в задачах моделирования

OpenCL (англ. Open Computing Language — открытый язык вычислений) — фреймворк для написания компьютерных программ, связанных с параллельными вычислениями на различных графических и центральных процессорах, а также FPGA. В OpenCL входят язык программирования, который основан на стандарте языка программирования Си C99, и интерфейс программирования приложений. OpenCL обеспечивает параллелизм на уровне инструкций и на уровне данных и является осуществлением техники GPGPU [1, 2].



Поддержка OpenCL. Для того чтобы проверить поддерживается ли OpenCL видеокартой загрузите GPU-Z [3]. При запуске программы

будет выведено окно, внизу первой вкладки которого отображены поддерживаемые технологии.

Установка. Сначала необходимо установить или обновить драйвер видеокарты [4–6]. OpenCL поставляется с различными инструментами разработки от Intel [7, 8], AMD, NVIDIA: Intel SDK for OpenCL Applications [9], oneAPI [10], CUDA-toolkit, ROCm [11].

OpenCL C++ Wrapper API. Оболочка C++ построена поверх OpenCL C API и не является его заменой. Реализация C++ Wrapper API вызывает базовый C API, который считается совместимой реализацией платформы спецификации OpenCL и API среды выполнения версии 1.2 или ниже. API C++ близко соответствует базовому API C и не требует дополнительных затрат на выполнение. Макросы заголовков C++ адаптируются к базовой версии C API, для которой компилируется заголовок. Интерфейс оболочки определяется в одном заголовочном файле C++ cl.hpp [12]. Некоторые инструменты разработки уже содержат cl.hpp, например, CUDA, но его так же можно загрузить здесь [13].

OpenCL C++ Bindings. Для многих крупных приложений предпочтительным языком является C++, поэтому кажется разумным определить привязки C++ для OpenCL. Интерфейс содержится в одном заголовочном файле C++ opencl.hpp, а все определения содержатся в пространстве имен cl. Нет дополнительных требований по включению cl.h и использованию привязки C++ или оригинального C. Достаточно просто подключить opencl.hpp. Сами привязки легковесны и близко соответствуют базовому C API. Использование привязок C++ не приводит к дополнительным затратам на выполнение. В новом заголовке есть многочисленные исправления совместимости, переносимости и управления памятью, а также дополнительные функции OpenCL 2.0. В результате заголовок не имеет прямой обратной совместимости, и по этой причине он выпущен как opencl.hpp, а не как новая версия cl.hpp[14, 15].

The C++ for OpenCL 1.0 and 2021.

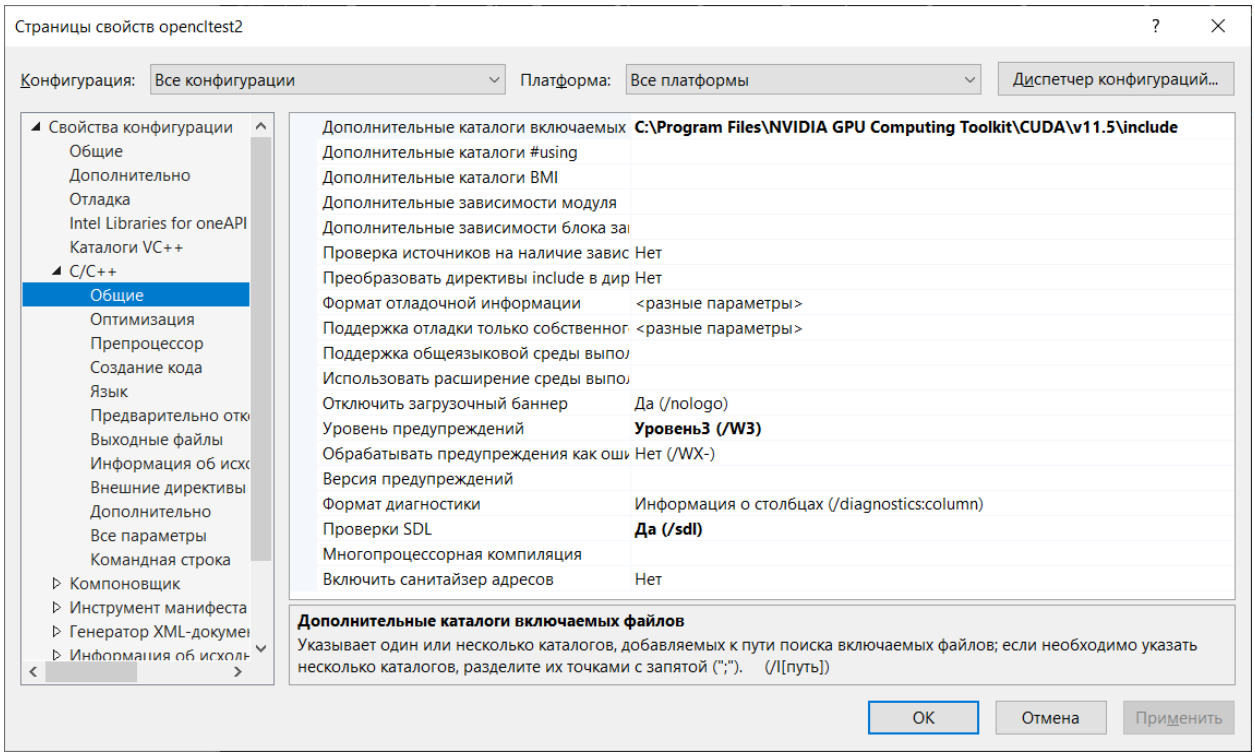
Этот язык построен на основе унифицированного OpenCL C 3.0 и C++17, что позволяет использовать большинство обычных функций

C++ в коде ядра OpenCL. Большая часть функций C++ и OpenCL C унаследована. Поскольку и OpenCL C, и C++ являются производными от C, и, кроме того, C++ почти полностью обратно совместим с C, основной принцип разработки C++ для OpenCL заключается в повторном применении существующих концепций OpenCL к C++ [16, 17].

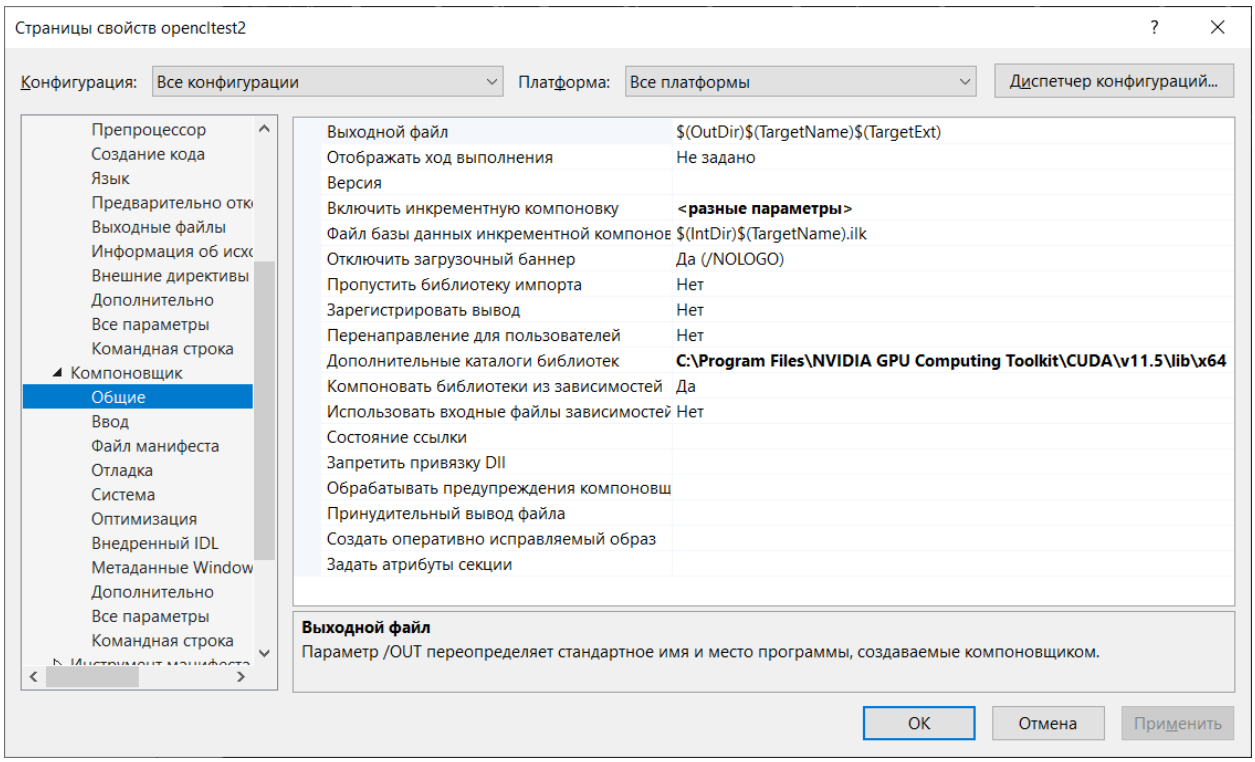
Ошибка: 'clCreateCommandQueue': объявлен deprecate. Функция clCreateCommandQueue устарела, начиная с OpenCL 2.0, и заменена на clCreateCommandQueueWithProperties [18]. Если вам нужно, чтобы ваш код работал на устройствах, которые еще не поддерживают OpenCL 2.0, вы можете продолжать использовать устаревшую функцию clCreateCommandQueue, используя макросы препроцессора, предоставляемые заголовками OpenCL, например:
#define CL_USE_DEPRECATED_OPENCL_1_2_APIS

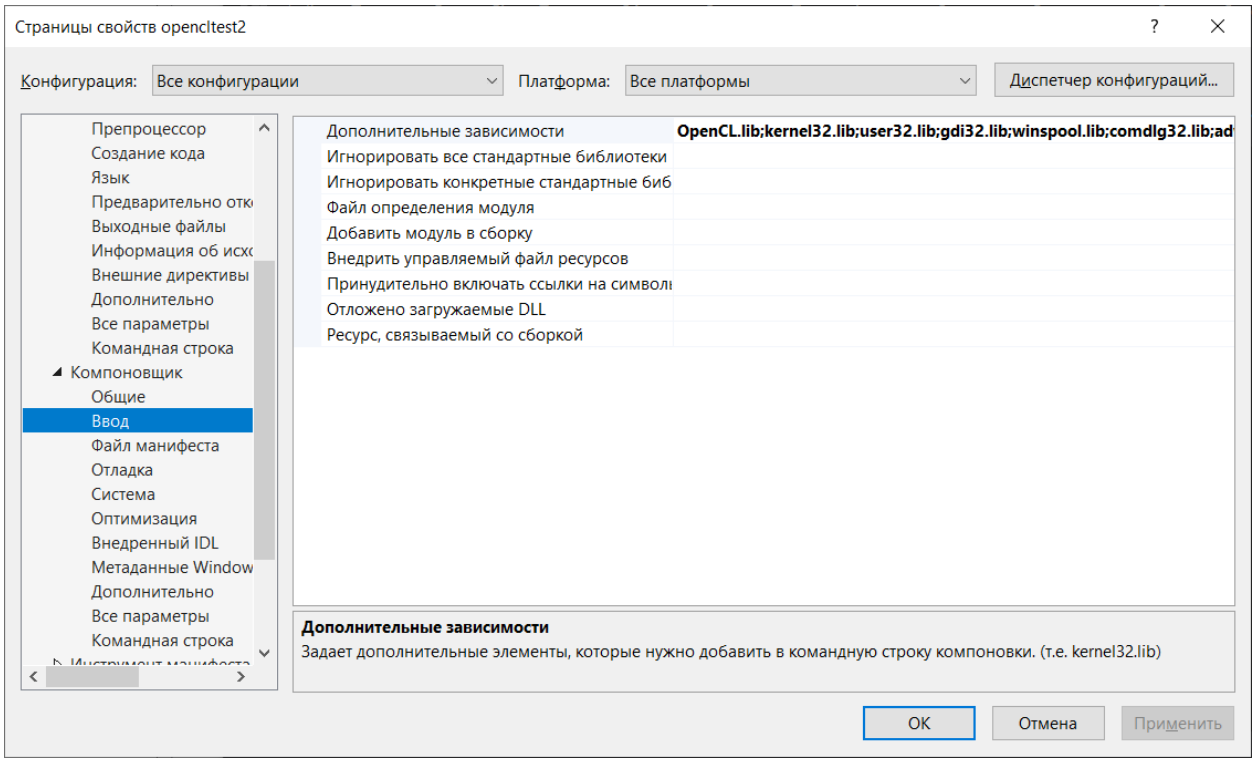
CUDA OpenCL. Так как установка *CUDA-toolkit* уже рассматривалась [19], сразу перейдем к настройке проекта. Создаем пустой проект C++. Далее необходимо указать путь до заголовочных файлов и файла библиотеки OpenCL. Примерные пути до файлов для *CUDA* выглядят так:
“C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.5\include\”;
“C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.5\lib\x64”.
Для *oneAPI*:
“C:\Program Files (x86)\Intel\oneAPI\compiler\2022.0.2\windows\include\sycl”;
“C:\Program Files (x86)\Intel\oneAPI\compiler\2022.0.2\windows\lib”.

Можно создать переменные среды и добавить расположение заголовочных файлов и библиотеки в них для удобства. Перейдем в свойства проекта и во вкладке *C/C++ → Общие* добавим расположение заголовочных файлов в строку *Дополнительные каталоги включаемых файлов*.



Во вкладке *Компоновщик → Общие* добавим путь к библиотеке в строке *Дополнительные каталоги библиотек*. А во вкладке *Компоновщик → Ввод* добавим *OpenCL.lib* в строку *Дополнительные зависимости*.





Создаем файл ядра *vector_add_kernel.cl* и вставляем в него код из листинга 1 [20]. Если у вас нет шаблона для *.cl* файла, можно создать, например, текстовый документ и переименовать его в *обозревателе решений*.

Листинг 1. Ядро для сложения векторов

```
__kernel void vector_add(__global const int* A,
__global const int* B, __global int* C) {

    // Получить индекс текущего элемента для
обработки
    int i = get_global_id(0);

    // Сложение векторов
    C[i] = A[i] + B[i];
}
```

Также создаем *.cpp* файл и вставляем в него код из листинга 2.

Листинг 2. Программа расчета суммы двух векторов

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
```

```
#ifdef __APPLE__
#include <OpenCL/opencl.h>
#else
#include <CL/cl.h>
#endif

#define MAX_SOURCE_SIZE (0x100000)

int main(void) {
    // создание двух векторов
    int i;
    const int LIST_SIZE = 1024;
    int* A = (int*)malloc(sizeof(int) *
LIST_SIZE);
    int* B = (int*)malloc(sizeof(int) *
LIST_SIZE);
    for (i = 0; i < LIST_SIZE; i++) {
        A[i] = i;
        B[i] = LIST_SIZE - i;
    }

    // чтение исходного кода ядра из
vector_add_kernel.cl
    FILE* fp;
    char* source_str;
    size_t source_size;

    fp = fopen("vector_add_kernel.cl", "r");
    if (!fp) {
        fprintf(stderr, "Failed to load
kernel.\n");
        exit(1);
    }
    source_str = (char*)malloc(MAX_SOURCE_SIZE);
```

```
    source_size = fread(source_str, 1,
MAX_SOURCE_SIZE, fp);
    fclose(fp);

    // Получение информации о платформах и
устройствах
    cl_platform_id platform_id = NULL;
    cl_device_id device_id = NULL;
    cl_uint ret_num_devices;
    cl_uint ret_num_platforms;
    cl_int ret = clGetPlatformIDs(1, &platform_id,
&ret_num_platforms);
    ret = clGetDeviceIDs(platform_id,
CL_DEVICE_TYPE_GPU, 1,
        &device_id, &ret_num_devices);
    // Создание OpenCL контекста
    cl_context context = clCreateContext(NULL, 1,
&device_id, NULL, NULL, &ret);

    // Создание очереди команд
    cl_command_queue command_queue =
clCreateCommandQueue(context, device_id, 0, &ret);

    // Создания буферов памяти на устройстве для
каждого вектора
    cl_mem a_mem_obj = clCreateBuffer(context,
CL_MEM_READ_ONLY,
        LIST_SIZE * sizeof(int), NULL, &ret);
    cl_mem b_mem_obj = clCreateBuffer(context,
CL_MEM_READ_ONLY, LIST_SIZE * sizeof(int), NULL,
&ret);
    cl_mem c_mem_obj = clCreateBuffer(context,
CL_MEM_WRITE_ONLY, LIST_SIZE * sizeof(int), NULL,
&ret);
```

```
    // Копирование векторов в буферы памяти
    ret = clEnqueueWriteBuffer(command_queue,
a_mem_obj, CL_TRUE, 0, LIST_SIZE * sizeof(int), A,
0, NULL, NULL);
    ret = clEnqueueWriteBuffer(command_queue,
b_mem_obj, CL_TRUE, 0, LIST_SIZE * sizeof(int), B,
0, NULL, NULL);

    // Создание программы из исходного кода ядра
    cl_program program =
clCreateProgramWithSource(context, 1, (const
char**)&source_str, (const size_t*)&source_size,
&ret);

    // Создание исполняемого файла
    ret = clBuildProgram(program, 1, &device_id,
NULL, NULL, NULL);

    // Создание OpenCL ядра
    cl_kernel kernel = clCreateKernel(program,
"vector_add", &ret);

    // Установка аргументов ядра
    ret = clSetKernelArg(kernel, 0,
sizeof(cl_mem), (void*)&a_mem_obj);
    ret = clSetKernelArg(kernel, 1,
sizeof(cl_mem), (void*)&b_mem_obj);
    ret = clSetKernelArg(kernel, 2,
sizeof(cl_mem), (void*)&c_mem_obj);

    // Выполнение ядра
    size_t global_item_size = LIST_SIZE;
    size_t local_item_size = 64;
```

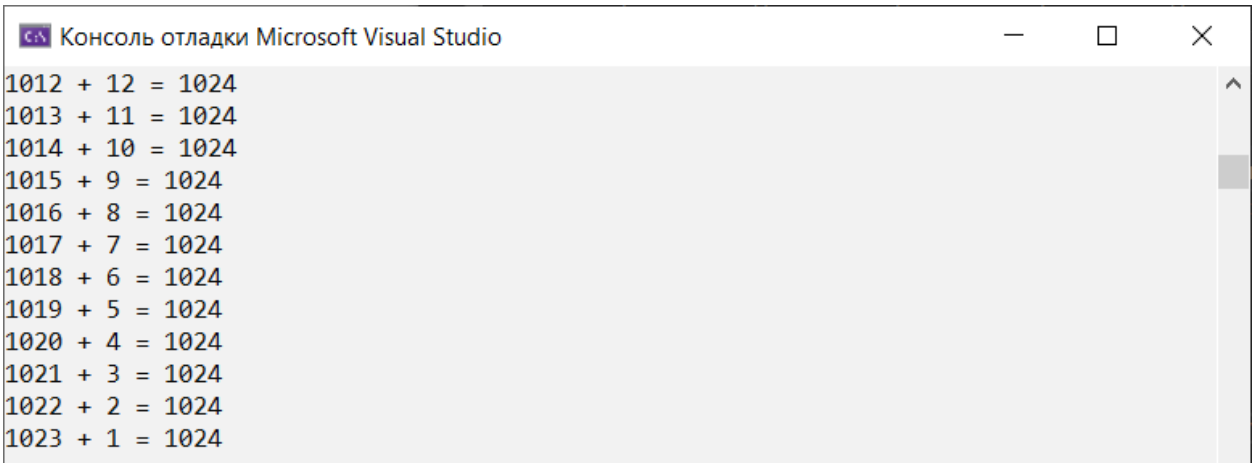
```
    ret = clEnqueueNDRangeKernel(command_queue,
kernel, 1, NULL, &global_item_size,
&local_item_size, 0, NULL, NULL);

    // Чтение результата из устройства в локальный
список C
    int* C = (int*)malloc(sizeof(int) *
LIST_SIZE);
    ret = clEnqueueReadBuffer(command_queue,
c_mem_obj, CL_TRUE, 0, LIST_SIZE * sizeof(int), C,
0, NULL, NULL);

    // Вывод результата на экран
    for (i = 0; i < LIST_SIZE; i++)
        printf("%d + %d = %d\n", A[i], B[i],
C[i]);

    // Отчистка памяти
    ret = clFlush(command_queue);
    ret = clFinish(command_queue);
    ret = clReleaseKernel(kernel);
    ret = clReleaseProgram(program);
    ret = clReleaseMemObject(a_mem_obj);
    ret = clReleaseMemObject(b_mem_obj);
    ret = clReleaseMemObject(c_mem_obj);
    ret = clReleaseCommandQueue(command_queue);
    ret = clReleaseContext(context);
    free(A);
    free(B);
    free(C);
    return 0;
}
```

Данная программа складывает два вектора.



Также можно ознакомиться с дополнительным материалом: OpenCL SDK [21], OpenCL Headers [22], OpenCL Resource Guide [23], начало работы с OpenCL [24].

http://www.kimrt.ru/index/course_stm/0-24

Проект реализуется победителем Конкурса на предоставление грантов преподавателям магистратуры 2020/2021 благотворительной программы «Стипендиальная программа Владимира Потанина» Благотворительного фонда Владимира Потанина.

Источники

1. OpenCL. URL: https://ru.wikipedia.org/wiki/OpenCL
2. OpenCL Оф. сайт. URL: https://www.khronos.org/opencl
3. GPU-Z. URL: https://www.techpowerup.com/gpuz/
4. Nvidia drivers. URL: https://www.nvidia.ru/Download/index.aspx?lang=ru
5. Intel drivers. URL:https://www.intel.ru/content/www/ru/ru/download-center/home.html
6. AMD drivers. URL: https://www.amd.com/ru/support
7. How Install the Intel OpenCL Driver, NVIDIA GPU Computing (CUDA) Toolkit. URL: https://www.youtube.com/watch?v=KUTVnxCeC50

8. Develop OpenCL Applications. URL: https://www.intel.com/content/www/us/en/developer/articles/tool/opencl-drivers.html#proc-graph-section
9. Intel SDK for OpenCL. URL: https://www.intel.com/content/www/us/en/developer/tools/opencl-sdk/overview.html
10. Создание проекта на языке DPC++. URL: http://www.kimrt.ru/courses/stm/self_work/DPC.pdf
11. ROCm. URL:https://www.amd.com/ru/graphics/servers-solutions-rocm
12. C++ wrappers. URL: https://www.khronos.org/registry/OpenCL/specs/opencl-cplusplus-1.2.pdf
13. C++ wrappers header. URL: https://github.com/KhronosGroup/OpenCL-Registry/tree/main/api/2.1
14. C++ bindings docs. URL: https://github.khronos.org/OpenCL-CLHPP/
15. C++ bindings. URL: https://github.com/KhronosGroup/OpenCL-CLHPP
16. The C++ for OpenCL 1.0 and 2021. URL: https://www.khronos.org/opencl/assets/CXX_for_OpenCL.html
17. cl_ext_cxx_for_opencl. URL: https://www.khronos.org/registry/OpenCL/extensions/ext/cl_ext_cxx_for_opencl.html
18. Ошибка: 'clCreateCommandQueue'. URL: https://stackoverflow.com/questions/28500496/opencl-function-found-deprecated-by-visual-studio
19. Установка CUDA toolkit. URL: http://www.kimrt.ru/courses/stm/self_work/Cuda_Toolkit.pdf
20. Getting started with OpenCL. URL: https://www.eriksmistad.no/getting-started-with-opencl-and-gpu-computing/

21. OpenCL SDK. URL: https://github.com/KhronosGroup/OpenCL-SDK/
22. OpenCL Headers. URL: https://github.com/KhronosGroup/OpenCL-Headers
23. Resourses map OpenCL. URL: https://www.khronos.org/opencl/resources
24. OpenCL. Как начать. URL: https://habr.com/ru/post/261323/