Казанский федеральный университет
ВЫСШАЯ ШКОЛА информационных технологий и информационных систем

Введение в OpenCL

Эдуард Храмченков

Open Compute Language

- Свободный программный фреймворк позволяющий писать параллельный код под различные платформы:
  - GPU любых производителей
  - Многоядерные CPU любых производителей
  - MIC
  - DSP/etc
- Один код для разного типа устройств
- Использует все ресурсы гетерогенных систем

Open Compute Language

- Стандарт разработан группой Khronos Compute
- Создан при поддержке Intel, AMD/ATI, Nvidia и Apple
- Зоопарк устройств и технологий – необходим единый интерфейс для работы с параллельными архитектурами
- Свободно распространяемый, бесплатный, кросс-платформенный фреймворк

Open Compute Language

- Область применения – от смартфона до суперкомпьютера
- Ускорение «тяжелых» задач на любой параллельной платформе
- Поддержка Embedded-систем
- Использование в системах дополненной реальности
- Параллельное программирование в авиации, автомобилях, смартфонах

Вендоры

Установка: OSX

- OpenCL работает из коробки
- Компиляция с ключом
  - *-framework OpenCL -DAPPLE*

Установка AMD GPU: Ubuntu

- Устновить пакеты
  - *sudo apt-get install build-essential linux-headers-generic debhelper dh-modaliases execstack dkms lib32gcc1 libc6-i386 opencl-headers*
- Скачать драйвера amd.com/drivers
- Создать установщик
  - *sudo sh fglrx*.run --buildpkg Ubuntu/precise*
- Установить драйвера
  - *sudo dpkg –i fglrx*.deb*
- Обновить xorg.conf
  - *sudo amdconfig --initial --adapter=all*
- Перезагрузка
- Проверка – запуск *fglrxinfo*

Установка AMD CPU: Ubuntu

- Скачать AMD APP SDK
- Распаковать архив
- Установка
  - *sudo ./Install*.sh*
- Создать линки
  - *sudo ln –s /opt/AMDAPP/lib/x86_64/* /usr/local/lib*
  - *sudo ln –s /opt/AMDAPP/include/* /usr/local/include*
- Обновить пути линковки
  - *sudo ldconfig*
- Перезагрузка и запуск команды *clinfo*

Установка Intel CPU: Ubuntu

- Скачать Intel® SDK for OpenCL™ Applications
- Распаковать архив
- Установить пакеты
  - *sudo apt-get install rpm alien libnuma1*
- Установка через alien
  - *sudo alien –i *base*.rpm *intel-cpu*.rpm *devel*.rpm*
- Скопировать ICD в папку
  - *sudo cp /opt/intel/<version>/etc/intel64.icd /etc/OpenCL/vendors/*

Установка Nvidia GPU: Ubuntu

- Внести открытый драйвер в черный список
  - sudo vim /etc/modprobe.d/blacklist.conf
  - blacklist nouveau
- Установить пакеты
  - *sudo apt-get install build-essential linux-header-generic opencl-headers*
- Скачать новый драйвер Nvidia
- Распаковать архив

Установка Nvidia GPU: Ubuntu

- В виртуальном терминале остановить оконный менеджер
  - *sudo service lightdm stop*
- Дать скрипту права на запуск и запустить
  - *chmod +x *.run*
  - *sudo ./*.run*
- Предустановочный тест может не пройти
- Ответить yes для DKMS, 32-bit GL libraries и на обновление X-config
- Перезагрузка

OpenCL Nvidia CUDA

- В Nvidia CUDA содержится OpenCL
- Компиляция с ключами:
  - *–I/путь к CUDA/include*
  - *–L/путь к CUDA/lib64*
  - *-lOpenCL*

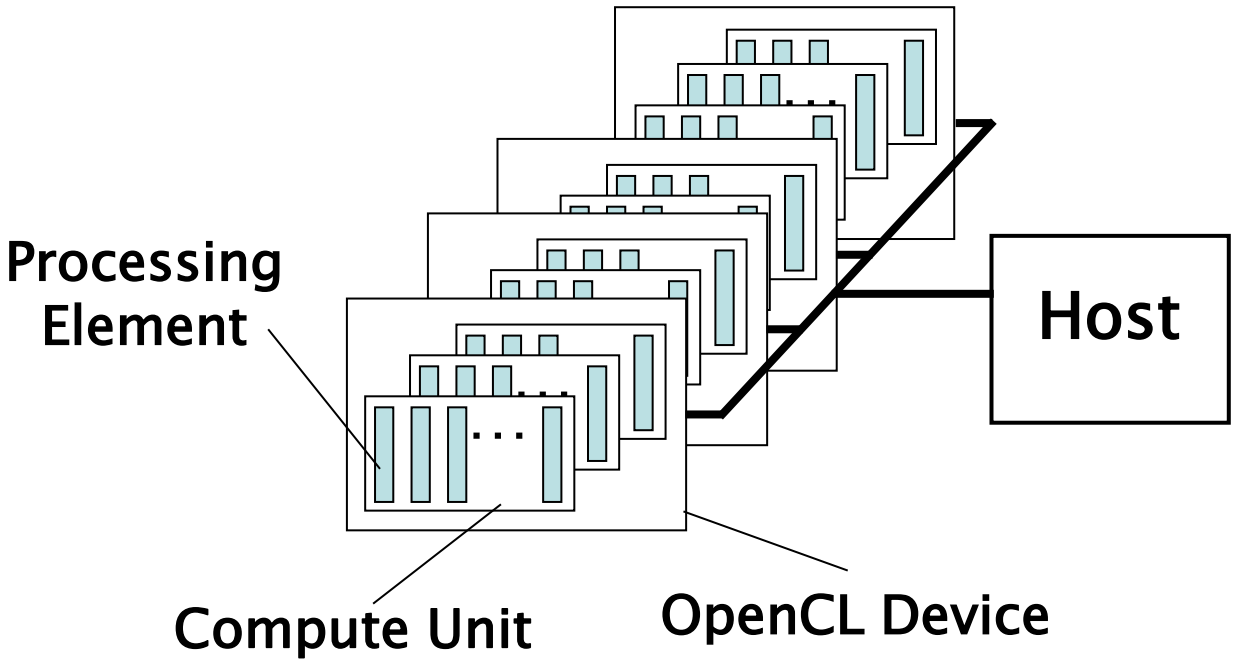
Пример 0

- Опрос всех доступных OpenCL устройств

Модель OpenCL

- Модель основана на концепциях **хоста(host)** и **устройств OpenCL (OpenCL Device)**
- Хост всегда один, устройств OpenCL может быть несколько
- Устройство OpenCL состоит из одного или нескольких **вычислительных узлов (Compute Unit, CU)**
- Каждый из вычислительных узлов состоит из **обрабатывающих элементов (Processing Element, PE)**
- Память делится на память хоста и память устройства

Модель OpenCL



Модель OpenCL

- Платформа OpenCL: 1 узел, 2 сокета CPU, 2 GPU
- CPU
  - Рассматриваются как одно устройство OpenCL
  - Один CU на ядро
  - Один PE на CU
- GPU
  - Каждый GPU – отдельное устройство OpenCL
  - Возможно одновременное использование всех CPU и GPU через OpenCL
- CPU будет являться собственным хостом

Исполнение OpenCL

- **Ядро**
  - Базовая единица исполняемого кода
  - Запускается на устройствах OpenCL
  - Параллелизм по данным и по заданиям
- **Хост код**
  - Исполняется на хосте
  - Отправляет ядра на исполнение устройствам OpenCL

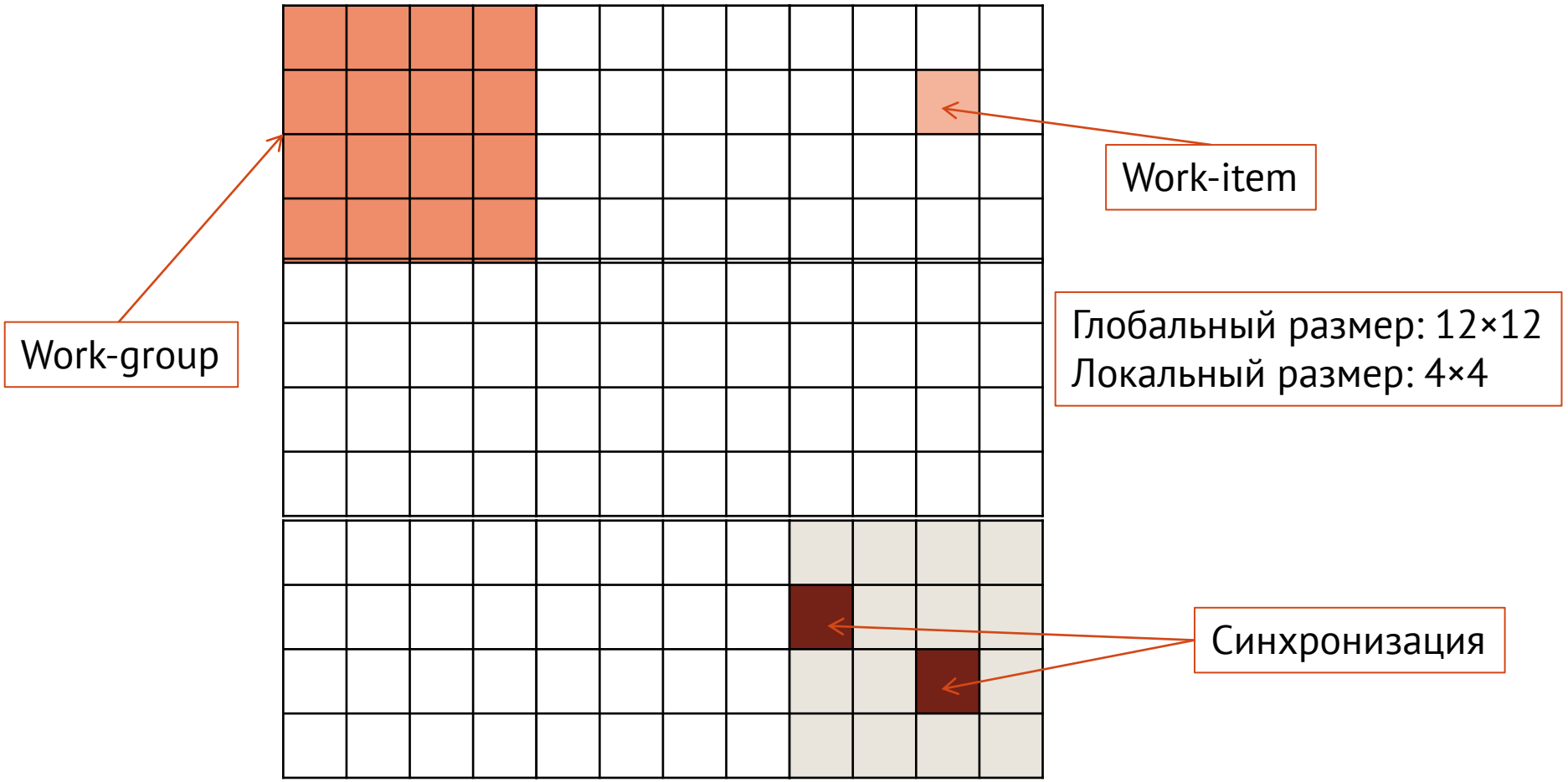
Исполнение OpenCL

- Ядро выполняется на сетке вычислений (Computation Domain) размерности n
- n = 1, 2, 3
- Каждый элемент сетки называется **рабочий элемент (work-item)**
- Сетка определяет общее число запущенных элементов – глобальные измерения
- Каждый рабочий элемент исполняет одно и то же ядро

Исполнение OpenCL

- Рабочие элементы объединены в **рабочие группы (work-group)**
- Локальные измерения определяют размерность рабочих групп
- Все элементы одной группы выполняются на одном и том же CU
- Общая разделяемая память и синхронизация между элементами
- Размер группы определяется пользователем или может быть задан автоматически

Исполнение OpenCL



Исполнение OpenCL

- Задача должна обладать «размерностью» – например, вычислить значение функции в каждой точке некоего куба
- Ядро запускается на 1,2,3-мерных сетках
- В каждом измерении мы определяем глобальный размер задачи
- Каждая точка исходной задачи обрабатывается своим рабочим элементом

Контекст OpenCL

- **Контекст** создается на хосте для управления ресурсами OpenCL
- Окружение в котором происходит исполнение ядер, синхронизация, управление памятью
- Контекст включает в себя
  - **Устройства – одно или несколько устройств OpenCL**
  - **Программные объекты – исходный или бинарный код реализующий ядра**
  - **Ядра – специальные функции для запуска на устройствах OpenCL**
  - **Объекты памяти – общие для хоста и устройств буферы памяти**

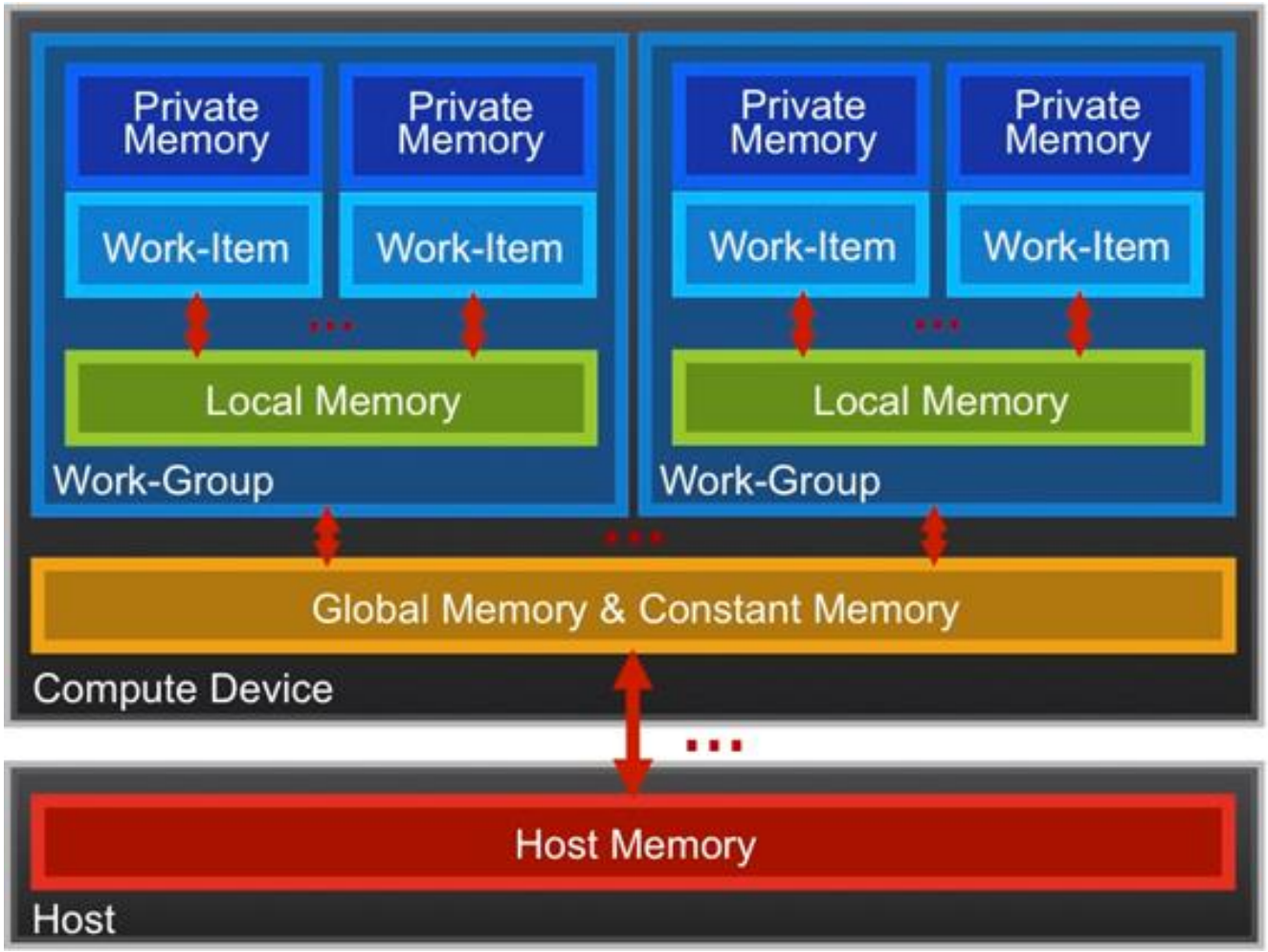
Очередь OpenCL

- Все команды устройству OpenCL от контекста поступают через **очередь команд**
- Очередь принимает
  - **Команды на запуск ядер**
  - **Операции передачи данных между хостом и устройством**
  - **Команды синхронизации**
- Каждая очередь команд соответствует отдельному устройству OpenCL
- Исполнение команд – по очереди/произвольно

Модель памяти

- Виды памяти
  - **Память хоста (Host)**
  - **Глобальная память (Global)** – доступна на чтение/запись всем рабочим элементам и группам и хосту
  - **Константная память (Constant)** – доступна на чтение/запись хосту, элементам только на чтение
  - **Локальная память (Local)** – используется для разделяемых данных в рабочей группе, доступна на чтение/запись элементам одной группы
  - **Частная память (Private)** – доступна элементу
- Управление памятью – вручную программистом

Модель памяти



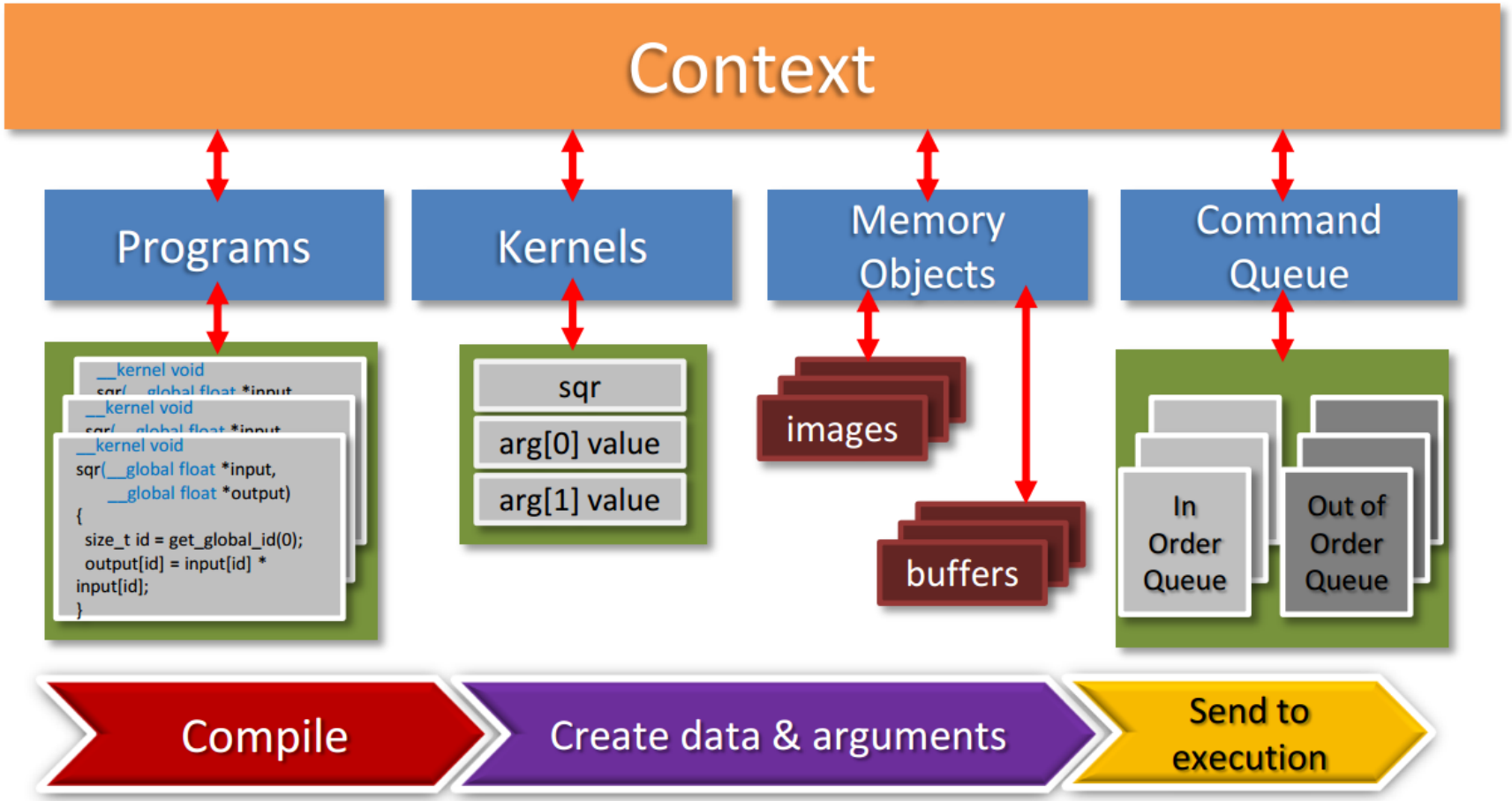
OpenCL vs Nvidia CUDA

OpenCL	Nvidia CUDA
Хост	Хост
Compute Unit	Stream Multiproccesor
Processing Element	Stream Processor
Глобальная память	Глобальная память
Константная память	Константная память
Локальная память	Разделяемая память
Частная память	Регистры

Приложение на OpenCL

- Любое приложение OpenCL состоит из
  - Кода хоста
  - Кода ядер
- Код хоста
  - Создает окружение для программы OpenCL
  - Создает ядра и управляет ими
  - Основные шаги
    - Определяет платформу – устройста, контексты и очереди
    - Создает программу – динамическую библиотеку ядра
    - Определяет параметры ядра
    - Отправляет ядро на исполнение и занимается передачей данных

Приложение на OpenCL



Платформа

- Получить первую доступную платформу OpenCL

```
cl_platform_id platforms;
cl_uint num_platforms;
cl_int err = clGetPlatfromIDs(
        1,//количество записываемых платформ
        &platforms,//список найденных OpenCL платформ
        &num_platforms//число доступных OpenCL платформ
);
```

Платформа

- Задействовать первый доступный GPU в данной платформе

```
cl_device_id device_id;
cl_uint num_of_devices;
err = clGetDeviceIDs(
        platforms, //номер платформы из clGetPlatformIDs
        CL_DEVICE_TYPE_GPU, //тип устройства
        1, //количество устройств добавляемых в список device_id
        &device_id, //список устройств
        &num_of_devices //количество найденных устройств
);
```

Платформа

- Поддерживаемые типы устройств OpenCL
  - CL_DEVICE_TYPE_CPU
  - CL_DEVICE_TYPE_GPU
  - CL_DEVICE_TYPE_ACCELERATOR
  - CL_DEVICE_TYPE_DEFAULT
  - Cl_DEVICE_TYPE_ALL

Платформа: контекст

- Создать простой контекст для одного устройства

```
cl_context context;
context = clCreateContext(
        platform_id, // номер платформы из clGetPlatformIDs
        1, //   количество устройств в списке device_id
        &device_id, // номер устройства
        NULL, // указатель на функцию обработки ошибок
        NULL, // аргументы функции обработки ошибок
        &err // код ошибки
);
```

Платформа: очередь

- Создать очередь команд ассоциированную с конкретным устройством

```
cl_command_queue command_queue;
command_queue = clCreateCommandQueue(
        context, // контекст OpenCL
        device_id, // устройство ассоциированное с контекстом
        0, // свойства
        &err // код ошибки
);
```

Программный объект

- **Программа** – набор функций-ядер
- **Функция** – написана на расширении OpenCL C
- **Функция-ядро** – имеет спецификатор __kernel
- **Программный объект (program object)** объединяет
  - Исходный или бинарный код программы
  - Последний успешно собранный исполняемый файл
  - Список устройств для которых собирался исполняемый файл
  - Опции и лог сборки
- Компиляция на этапе выполнения

Программный объект

- Создать программный объект из исходников

```
const char *ProgramSource =
"__kernel void hello(__global float *input, __global float *output)\n"\
"{\n"\
" size_t id = get_global_id(0);\n"\
" output[id] = input[id] * input[id];\n"\
"}\n";
cl_program program;
program = clCreateProgramWithSource(
        context, // контекст
        1, // количество строк в следующем параметре
        (const char **) &ProgramSource, // массив строк
        NULL, // длина каждой строки или нуль-терминированная строка
        &err // код ошибки
);
```

Сборка программы

- Компиляция и линковка программного объекта
- Подобие «динамической библиотеки» из которой потом вызываются нужные функции-ядра

```
err = clBuildProgram(
        program, // программный объект
        0, // количество устройств в списке устройств
        NULL, // список устройств – NULL значит для всех устройств
        NULL, // опции сборки
        NULL, // функция обратного вызова после компиляции
        NULL // аргументы для функция обратного вызова
);
```

Объект памяти

- Объект памяти – указатель (handle) на регион глобальной памяти устройства
- Два типа объектов
  - Буфер
    - С-подобный массив, линейная последовательность байтов
    - Доступны ядрам через указатели
  - Изображение
    - Задает двух- и трехмерные регионы памяти
    - Доступны на чтение/запись только при помощи специальных функций
    - Применяются при совместной работе с графическими API

Объект памяти

- Объекты памяти создаются с хоста при помощи специальных функций

```
cl_mem input;
input = clCreateBuffer(
        context, // контекст
        CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR//флаги
        sizeof(type) * DATA_SIZE, // размер данных
        inputsrc, // указатель на память хоста, откуда копировать данные
        &err // код ощибки
);
```

- Флаги задают использование памяти, например:
  - Только для чтения | Данные скопировать с хоста

Объект памяти

- Запись данных в буфер на устройстве из памяти хоста

```
err = clEnqueueWriteBuffer(
        command_queue, // очередь
        input, // указатель на буфер, куда производится запись
        CL_TRUE, // задает блокирующую запись
        0, // отступ в байтах в записываемом буфере
        sizeof(float) *DATA_SIZE, // размер считываемых данных
        host_ptr, // указатель на память хоста, откуда копируются данные
        NULL, // количество событий в списке событий
        NULL, // события, которые надо завершить до копирования
        NULL // объект события, который нужно вернуть при завершении
);
```

Объект памяти

- Запись данных из буфера на устройстве в память хоста

```
err = clEnqueueReadBuffer(
        command_queue, // очередь
        input, // указатель на буфер, куда производится запись
        CL_TRUE, // задает блокирующую запись
        0, // отступ в байтах в записываемом буфере
        sizeof(float) *DATA_SIZE, // размер считываемых данных
        host_ptr, // указатель на память хоста, откуда копируются данные
        NULL, // количество событий в списке событий
        NULL, // события, которые надо завершить до копирования
        NULL // объект события, который нужно вернуть при завершении
);
```

Объект ядра

- **Объект ядра** – инкапсулирует указанную функцию-ядро вместе с аргументами
- Объект ядра посылается через очередь на исполнение

```
cl_kernel kernel;
kernel = clCreateKernel(
        program, // программный объект
        "hello", // имя ядра объявленного со спецификатором __kernel
        &err // код ошибки
);
```

Объект ядра

- Требуется присоединить к объекту ядра все аргументы соответствующей __*kernel*-функции
- Аргументы в функции-ядре должны объявляться со спецификаторами __*global* или __*constant*

```
err = clSetKernelArg(
        kernel, // объект ядра
        0, // индекс аргумента
        sizeof(cl_mem), // размер аргумента
        &input_data // указатель на данные аргумента
);
```

Постановка в очередь

- Объект ядра необходимо поставить в очередь на выполнение

```
err = clEnqueueNDRangeKernel(
        command_queue, // очередь
        kernel, // объект ядра
        1, // количество измерений задачи
        NULL, // зарезервированный параметр, д.б. NULL
        &global, // количество рабочих элементов
        &local, // количество рабочих групп
        NULL, // количество событий в списке событий
        NULL, // события, которые надо завершить до копирования
        NULL // объект, который нужно вернуть при завершении
);
```

Очистка данных

- После завершения необходимо освободить память и уничтожить объекты

```
clReleaseMemObject(input);
clReleaseMemObject(output);
clReleaseProgram(program);
clReleaseKernel(kernel);
clReleaseCommandQueue(command_queue);
clReleaseContext(context);
```

Пример 1

- Вычисление квадратов элементов вектора

Казанский федеральный
УНИВЕРСИТЕТ
ВЫСШАЯ ШКОЛА
информационных технологий
и информационных систем

Вопросы

ekhramch@kpfu.ru