2. Стандарт программирования гетерогенных вычислений OpenCL

Использованные источники

Программирование на OpenCL. Бастраков С.И. ВМК ННГУ
www.hpcc.unn.ru/file.php?id=565

Стандарт **OpenCL**

OpenCL (*Open Computing Language* — открытый язык вычислений) – открытый стандарт для гетерогенных вычислений, разрабатываемый некоммерческим консорциумом *Khronos Group* совместно с представителями производителей устройств и программного обеспечения. Создан в 2008-2009 гг.

Стандарт допускает программирование для:

1. ЦПУ (*CPU*);
2. Графических процессоров (*GPU*);
3. Программируемых логических матриц (*FPGA*).

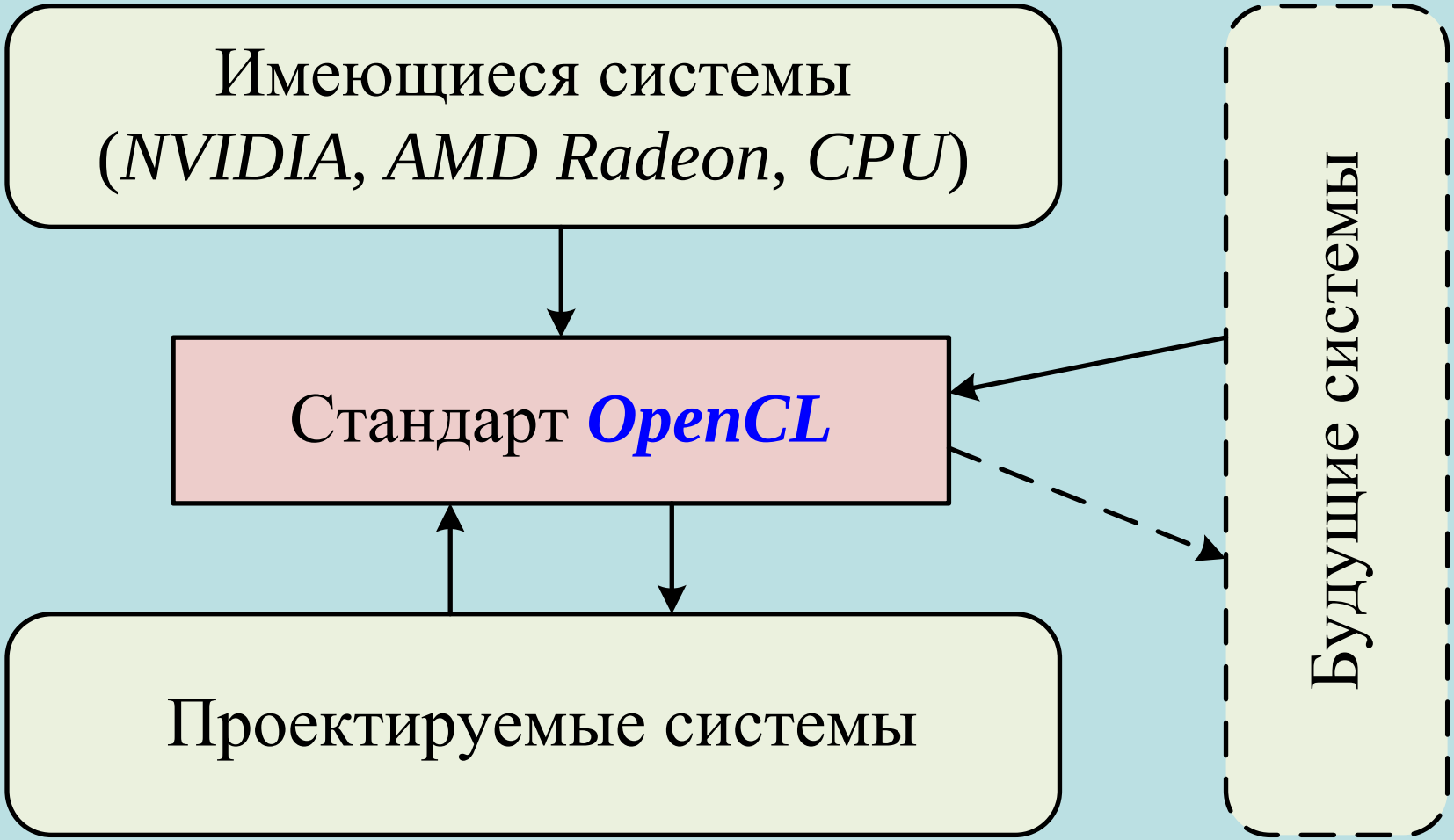
Khronos Group

Khronos Group — промышленный консорциум, целью которого является выработка открытых стандартов интерфейсов программирования (*API*) в области создания и воспроизведения динамической графики и звука на широком спектре платформ и устройств, с поддержкой аппаратного ускорения. В консорциум входят более 100 компаний (в т.ч. *Apple*, *NVIDIA*, *AMD*, *Intel*). [*wikipedia.org*]

Создатель *OpenGL*, *OpenML*, *OpenWG*, *OpenMAX*, *OpenCL* и пр.

Сайт: *www.khronos.org*

Стандарт **OpenCL**



Основные особенности OpenCL

1. Исходный программный код легко переносится с одной платформы на другую.
2. Поддержка широкого класса устройств достигается за счет введения обобщенных моделей данных систем:

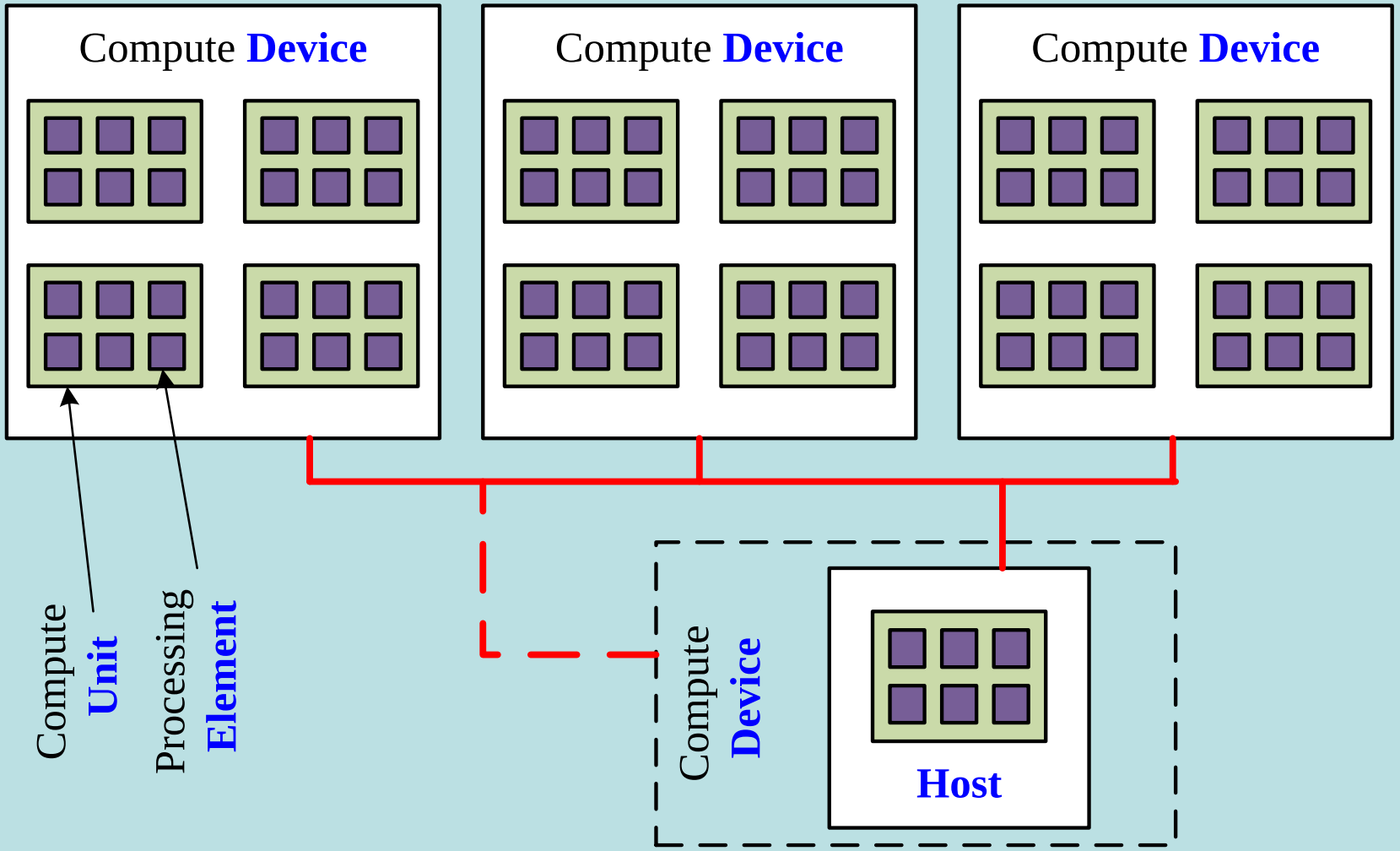
– **модель платформы** (*platform model*);
– **модель исполнения** (*execution model*);
– **модель памяти** (*memory model*);
– **модель программирования** (*programming model*).

3. Все модели являются абстрактными (не привязанными к конкретным параметрам устройств), реализация предоставляется производителем.

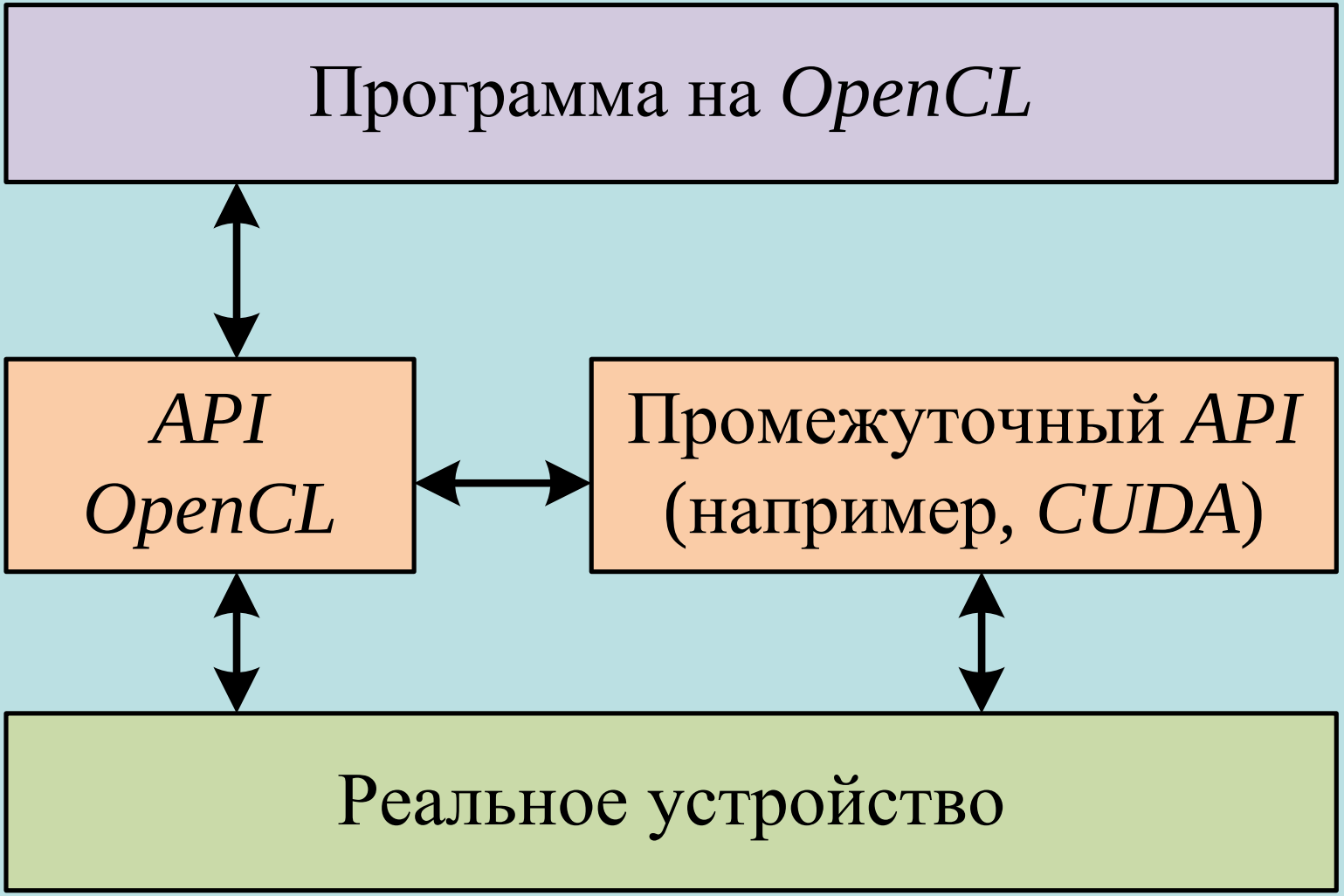
Platform model

1. Платформа представляется в виде хост-системы (*host*), связанной с одним или несколькими устройствами (*device*).
2. Центральный процессор может являться одновременно и хост-системой и устройством.
3. Устройство состоит из одного или более вычислительных модулей (*compute units*), которые могут включать в себя несколько обрабатывающих элементов (*processing elements*).
4. Непосредственно вычисления производятся в обрабатывающих элементах устройства.

Platform model



API OpenCL


Программа на *OpenCL*
API OpenCL
Промежуточный *API* (например, *CUDA*)
Реальное устройство

Получение списка всех доступных платформ

cl_int **clGetPlatformIDs**

(cl_uint **num_entries*, cl_platform_id *platforms*, cl_uint **num_platforms*)

Узнать параметры каждой платформы можно с помощью функции

clGetPlatformInfo

Получение списка всех устройств заданного типа

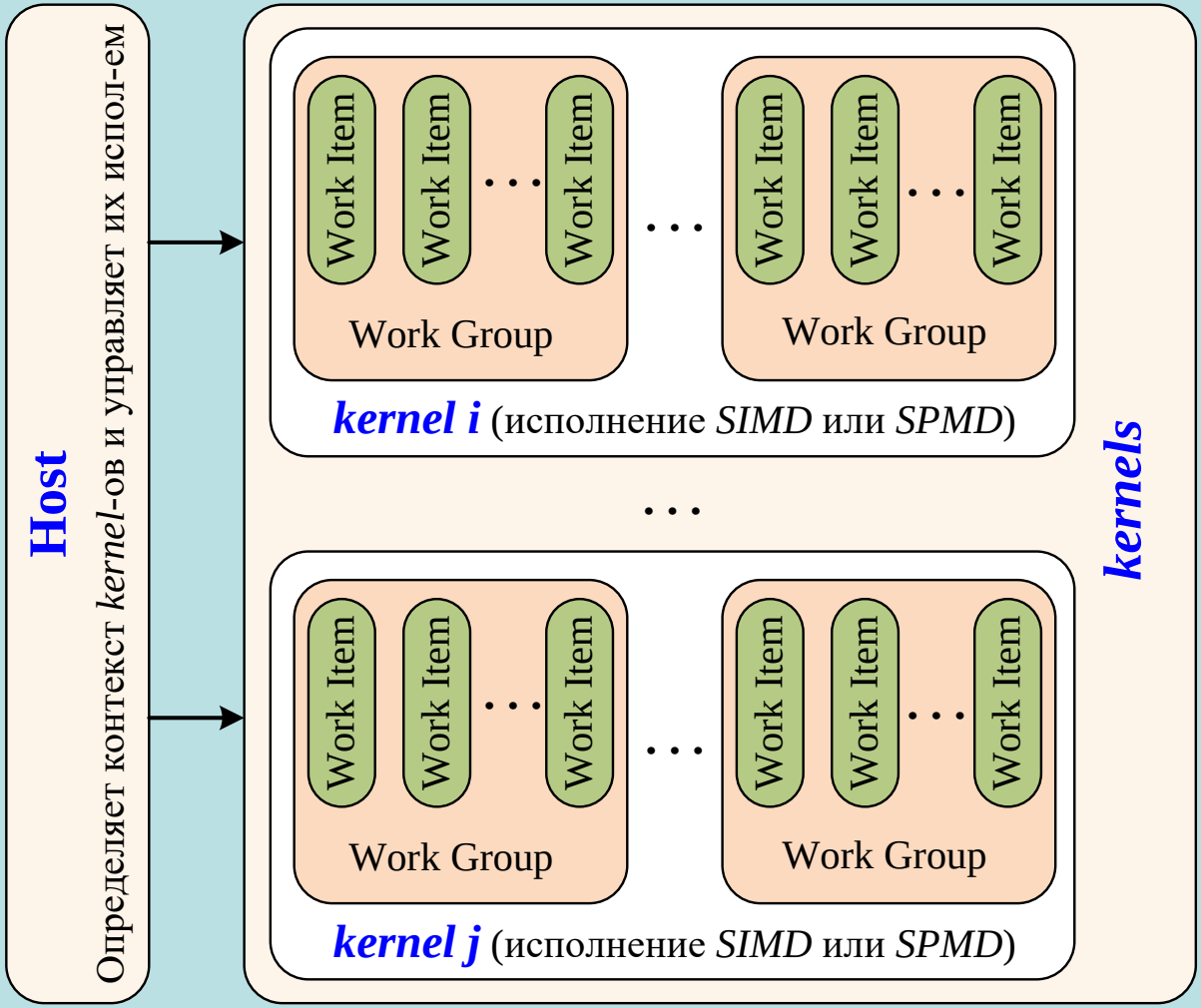
cl_int **clGetDeviceIDs**

(cl_platform_id *platform*, cl_device_type **device_type*, cl_uint **num_entries*, cl_device_id *devices*, cl_uint **num_devices*)

Узнать параметры устройства можно с помощью функции

clGetDiviceInfo

Execution model



Execution model
контекст исполнения

Контекст исполнения включает в себя:

- **Устройства (Compute Devices):** набор *OpenCL*-устройств.
- **Ядра (Kernels):** *OpenCL* функции, которые исполняются на устройствах.
- **Объекты программ (Program Objects):** исходные коды и исполняемые файлы kernels.
- **Объекты памяти (Memory Objects):** набор объектов в памяти, видимых как хосту, так и *OpenCL* устройству (в.т.ч. *kernel*-ам).

Создание контекста

cl_context **clCreateContext**

(const cl_context_properties **properties*, cl_unit *num_devices*, void **pfn_notify*, void **user_data*, cl_int **ercode_ret*);

Очередь команд

Очередь команд (***command queue***) является механизмом запроса действия на устройстве со стороны хоста. В качестве действия на устройстве могут выступать **операции с памятью**, **запуск ядер**, **синхронизация**. Для каждого устройства требуется своя очередь команд. Команды внутри очереди могут выполняться синхронно и асинхронно; в порядке установки или нет.

Создание очереди команд

cl_command_queue **clCreateCommandQueue**

(cl_context *context*, cl_device_id *device*,

cl_command_queue_properties **properties*, cl_int **ercode_ret*);

Объекты памяти

- Все операции работы с памятью на устройстве осуществляются с использованием **объектов памяти**.
- Прямая работа с памятью устройства со стороны хоста невозможна (даже если это *CPU*).
- Для представления одномерных массивов данных используются **буферы** (*buffer objects*). Данные представлены в непрерывном участке памяти, есть прямой доступ со стороны устройства как к массивам.
- Для представления 2- и 3-мерных массивов данных используются **изображения** (*image objects*). Для доступа со стороны устройства используются специальные объекты – **сэмплеры** (*sampler objects*).

Создание буфера

cl_mem **clCreateBuffer**

(cl_context *context*……., cl_int **ercode_ret*);

Передача данных между Host и Compute Unit

Функции для передачи данных:

clEnqueue{Read|Write}{Buffer|Image}

Установка прямого соответствия между участками памяти на *host* и *compute unit*

clEnqueueMap{Buffer|Image}

Создание объекта программы

Объект программы (*program object*) служит для представления следующих данных: исходные и/или скомпилированные тексты ядер; данные о компиляции.

cl_program **clCreateProgramWithSource**

(cl_context *context*, const char ***strings*, const size_t **lengths*, cl_int **ercode_ret*); // объект из исходного текста

Создание объекта ядра

cl_kernel **clCreateKernel**

(cl_program *program*, const char **kernel_name*, cl_int **ercode_ret*)

// создаёт объект ядро для имеющейся в тексте программы функции с заданным именем

Компиляция программы

cl_int **clBuildProgram** (cl_program *program*, cl_unit *num_devices*, cl_device_id *device*, const char **options*, void **pfn_notify*, void **user_data*);

Установка параметров ядра

cl_int **clSetKernelArg** (cl_kernel *kernel*, cl_unit *arg_index*, size_t *arg_size*, const void **arg_value*)

// Вызывается для каждого параметра ядра

Запуск ядра

cl_int **clEqueueNDRangeKernel** (cl_command_queue *command_queue*, cl_kernel *kernel*, cl_unit *work_dim*, const size_t **global_work_offset*, const size_t **global_work_size*, const size_t **local_work_size*, cl_unit *num_events_in_wait_list*, const cl_event **event_wait_list*, cl_event **event*);

Execution model индексация

Пространство индексов в *OpenCL* называется **NDRange**.

Каждый Work Item и каждая Work Group может индексироваться 1-, 2- и 3-мерным индексом.

NDRange – массив целых чисел (*integer*) длины *N*, указывающий размерность в каждом из направлений.

Execution model индексация

Получение индексов осуществляется с помощью функций:

get_global_id(dim) - глобальный id

get_global_size(dim)

get_group_id(dim) – id группы

get_num_groups(dim)

get_local_id(dim) - глобальный id в группе

get_local_size(dim)

где dim = **0**, **1** или **2** - размерность

Сравнение ядер

Код ядра на CUDA

```
__global__ void vectorAdd(const float * a, const float * b, float * c)
{
int nIndex = blockIdx.x * blockDim.x + threadIdx.x;
c[nIndex] = a[nIndex] + b[nIndex];
}
```

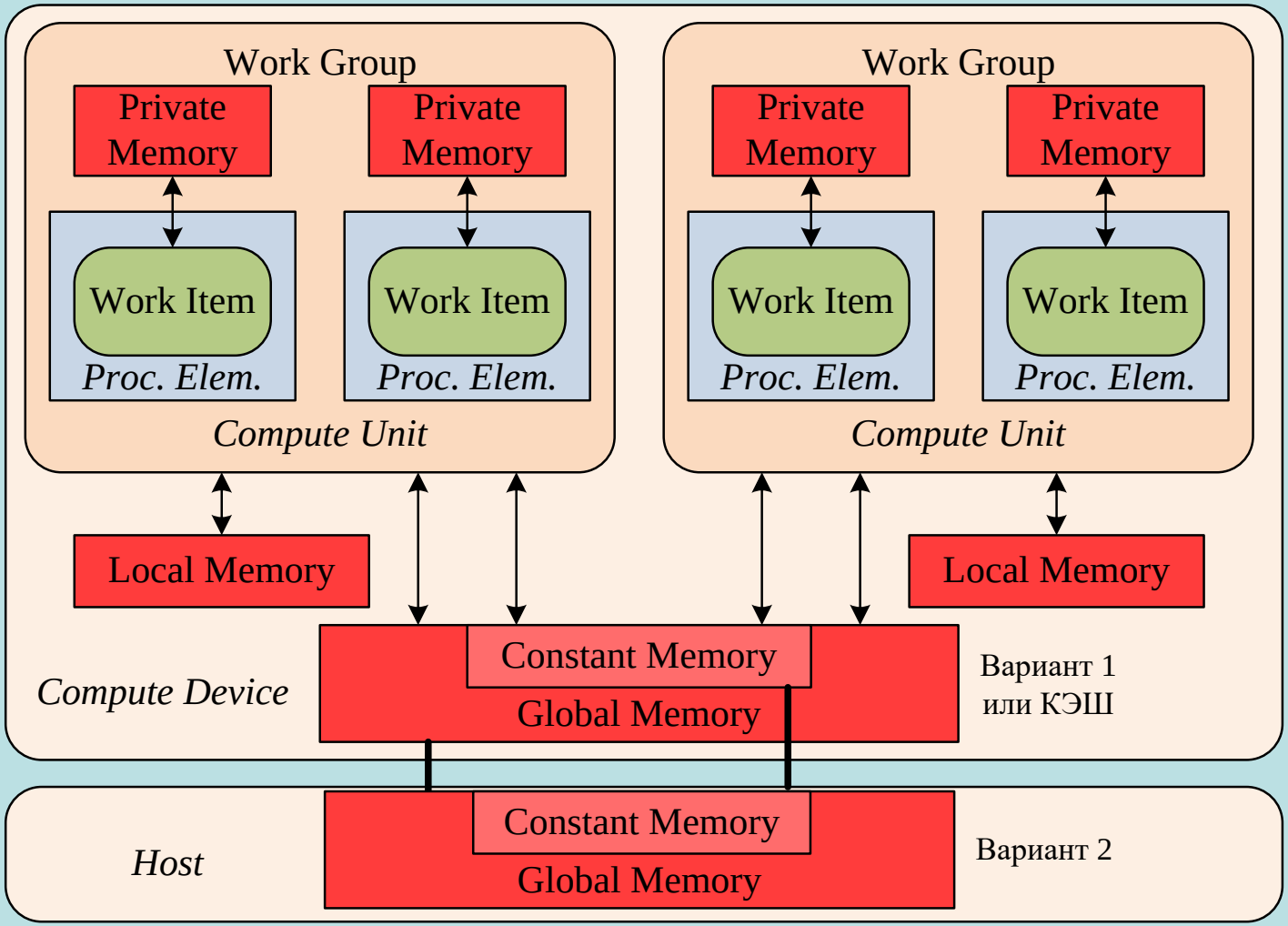
Код ядра на OpenCL

```
__kernel void vectorAdd(__global const float * a, __global const float * b, __global float * c)
{
int nIndex = get_global_id(0);
c[nIndex] = a[nIndex] + b[nIndex];
 }
```

Memory model

1. **Глобальная память.** Эта память предоставляет доступ на чтение и запись элементам всех групп. Каждый *Work-Item* может писать и читать из любой части объекта памяти.
2. **Константная память.** Область глобальной памяти, которая остается постоянной во время исполнения *kernel*'а. Хост аллоцирует и инициализирует объекты памяти, расположенные в константной памяти.
3. **Локальная память.** Область памяти, локальная для группы. Эта область памяти может использоваться, чтобы хранить переменные, доступные всей группе.
4. **Частная (*private*) память.** Область памяти, принадлежащая Work-Item.

Memory model



Memory model
Квалификаторы данных

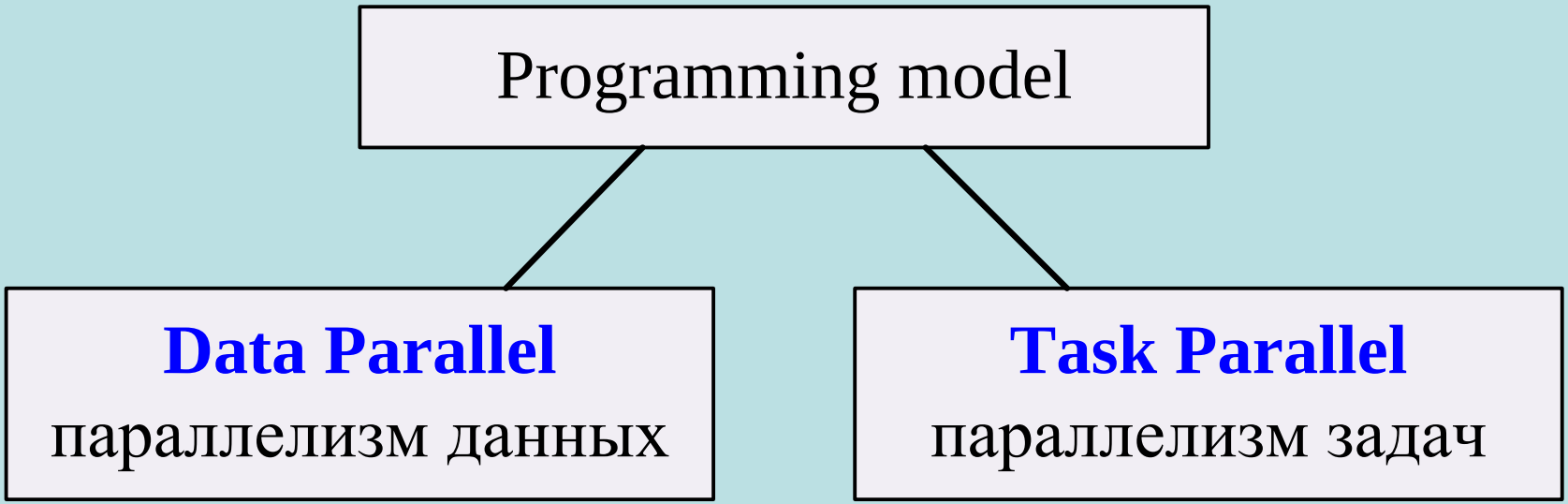
__**global** или **global** – данные в глобальной памяти.

__**constant** или **constant** – данные в константной памяти.

__**local** или **local** – данные в локальной памяти.

__**private** или **private** – данные в частной памяти.

Programing model



Programing model

Параллелизм по данным (*data parallel*):

Каждый *Work Item* выполняет фиксированное набор однотипных операций, масштабируется количество *Work Item* и *Work Group*.

Параллелизм по задачам (*task parallel*):

–Разные *kernel* исполняются независимо на различных пространствах индексов.

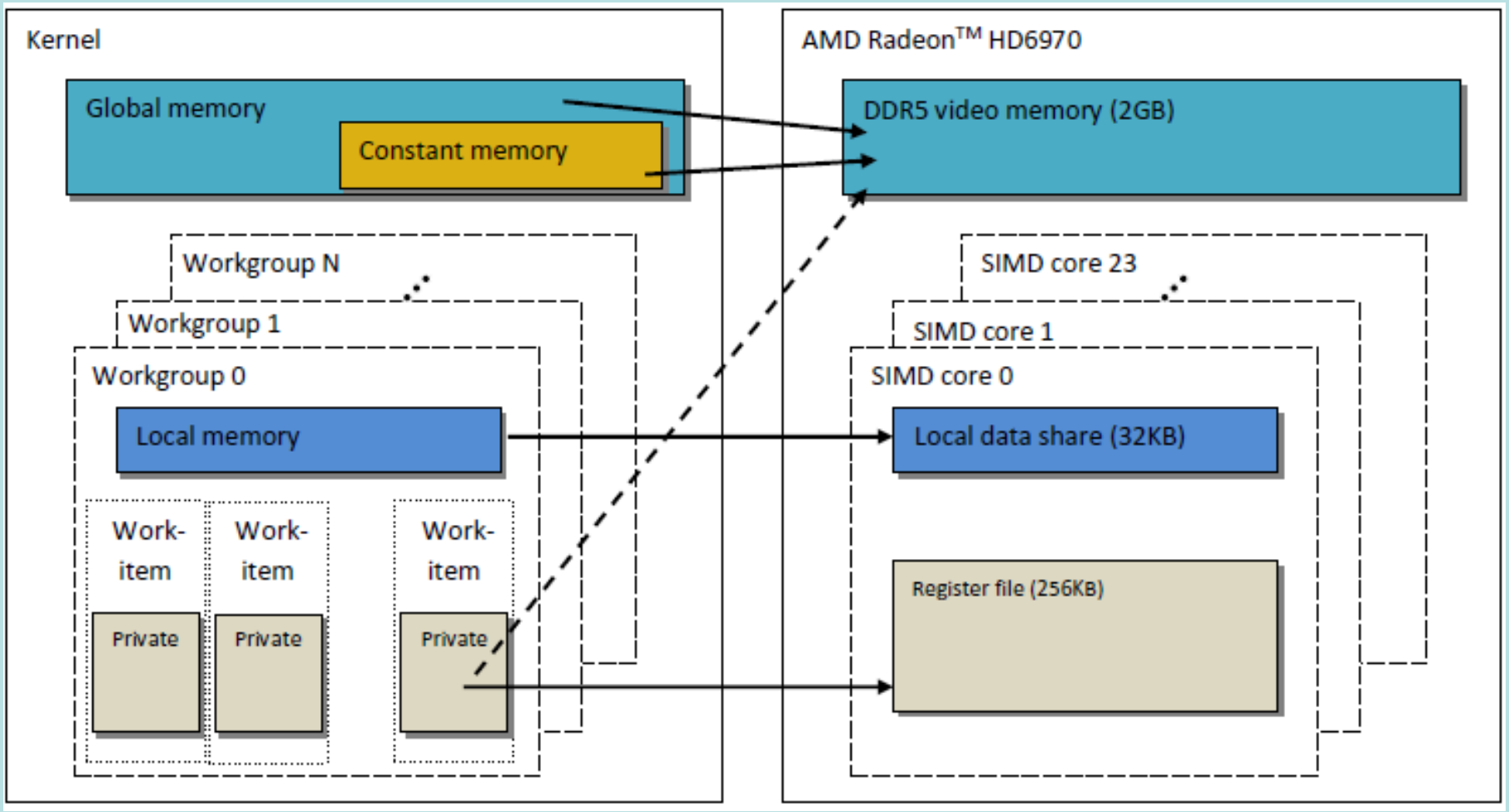
–Постановка в очередь нескольких задач.

Синхронизация*:*

–Между *Work Item* в одной *Work Group*.

–Между командами в одной очереди команд.

Пример из интернета




2
Выбор устр-ва
3
Включ. устр-ва
В очередь команд
Файл с исх.
текстом ядра
5
Созд. объект.
ядра
Созд. Объект.
память
1
Созд. объект
контекст
4
6
Созд. объекта
программы
7
Сборка
программы
9
Установка входных
параметров
11
Постановка
ядра в очередь
10
Копирование данных
на устрйство
8
Создание исп.
ядра
12
Чтение
результатов

Этапы выполнения задачи

1. Просмотр имеющихся платформ и устройств, получение их идентификаторов
2. Создание контекст для исполнения нашей программы на устройстве.
3. Выбор необходимого устройства (можно сразу выбрать устройство с наибольшим количеством *Flops*).
4. Инициализация выбранного устройства созданным нами контекстом.
5. Создание очереди команд на основе *ID* устройства и контекста.

Этапы выполнения задачи

6. Создаем программу на основе исходных кодов и контекста,
   либо на основе бинарных файлов и контекста.
7. Сборка программы (*build*).
8. Создание *kernel*.
9. Создание объектов памяти для входных и выходных данных.
10. Постановка в очередь команд записи данных из области памяти с данными на хосте в память устройства.

Этапы выполнения задачи

11. Постановка в очередь команды исполнения созданного нами *kernel*.
12. Постановка в очередь команды считывания данных из устройства.
13. Ожидание завершения операций.