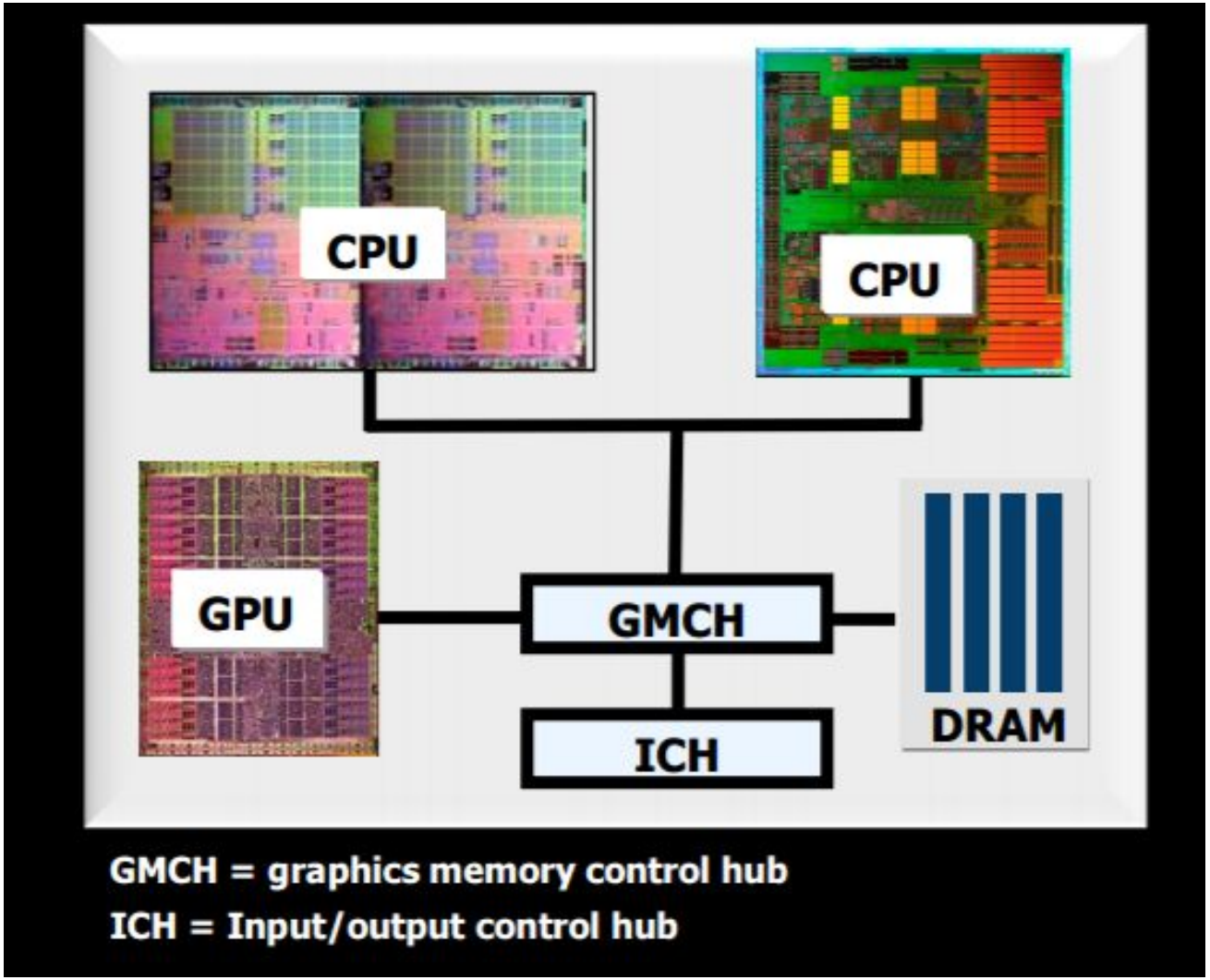
Лекция 8. OpenCL

Соснин В.В. Балакшин П.В.

Материалы этой презентации взяты в том числе из книг

1. «Introduction to OpenCL Programming». – AMD, 2010.
2. «Introduction to OpenCL». – Nvidia, 2011

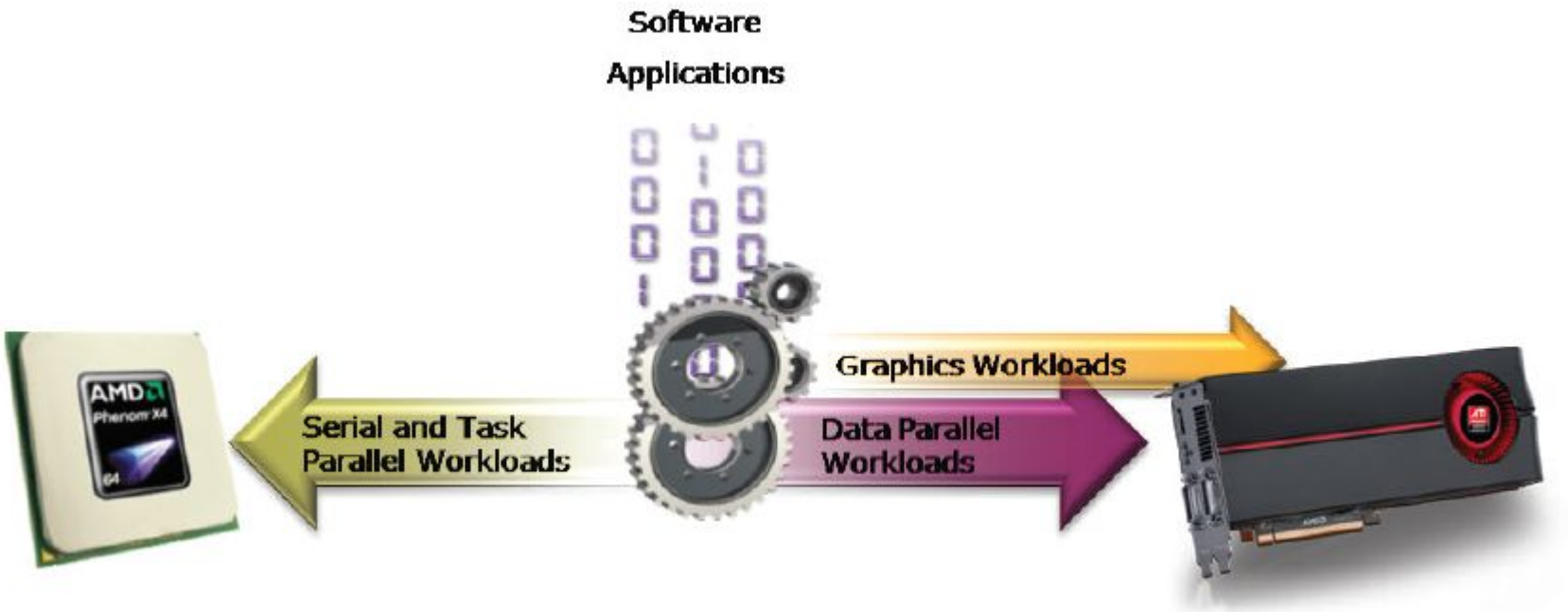
Гетерогенные vs гомогенные параллельные вычисления



Что такое OpenCL?

- OpenCL (от англ. Open Computing Language — открытый язык вычислений) — фреймворк для написания компьютерных программ, связанных с параллельными вычислениями на различных графических (англ. GPU) и центральных процессорах (англ. CPU).
- Цель OpenCL - дополнить OpenGL и OpenAL, которые являются открытыми отраслевыми стандартами для трёхмерной компьютерной графики и звука, пользуясь возможностями GPU.
- Консорциум Khronos Group, в который входят много крупных компаний, включая Apple, AMD, Intel, nVidia, ARM, Sun Microsystems, Sony Computer Entertainment и другие.
- Первая версия стандарта – ноябрь 2008 г.
- Текущая документация - https://www.khronos.org/registry/OpenCL/specs/opencl-2.2.pdf
- Полезная ссылка: http://docplayer.ru/37490743-Programmirovanie-na-opencl.html

Типовая модель использования OpenCL



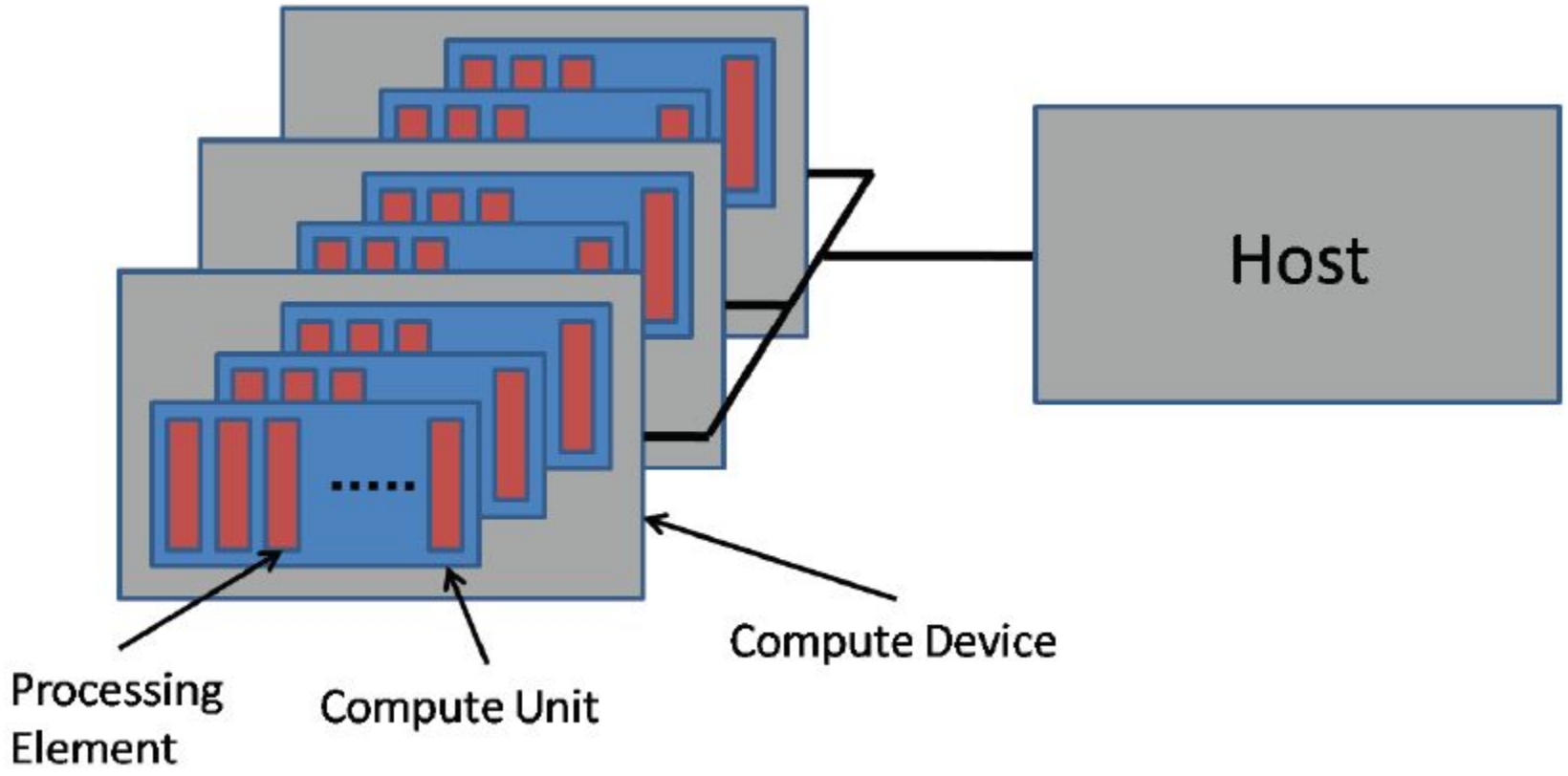
Распараллеливание по задачам (единицы/десятки сложных производительных ядер)

Распараллеливание по данным (тысячи простых медленных ядер).

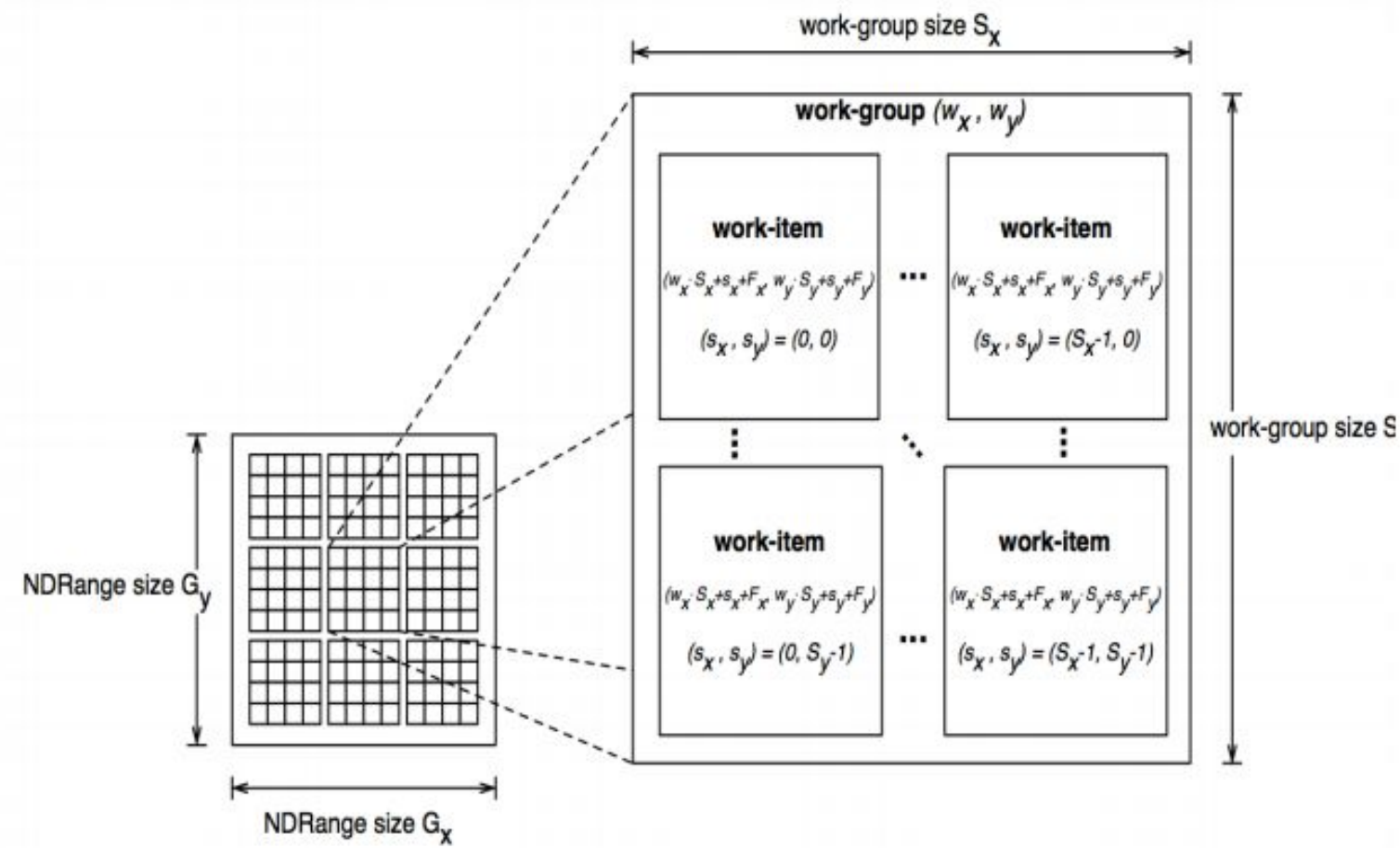
Язык программирования в OpenCL

- Derived from ISO C99 (with some restrictions)
- Language Features Added
  - Work-items and work-groups
  - Vector types
  - Synchronization
  - Address space qualifiers
- Also includes a large set of built-in functions
  - Image manipulation
  - Work-item manipulation
  - Math functions

Как OpenCL видит аппаратуру (платформу)

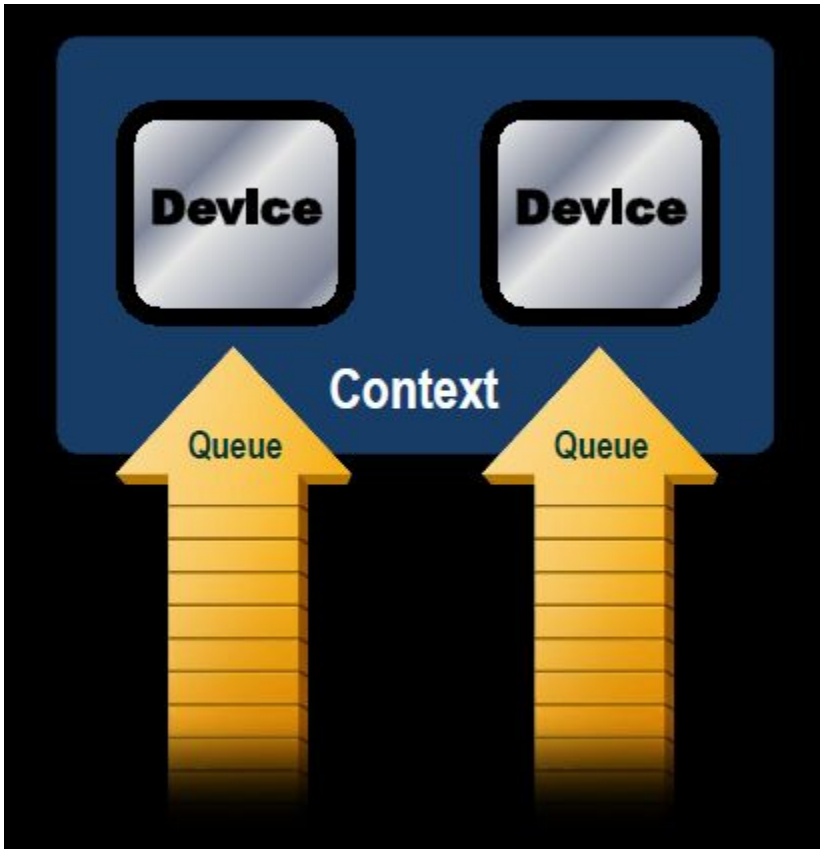


Принцип работы OpenCL



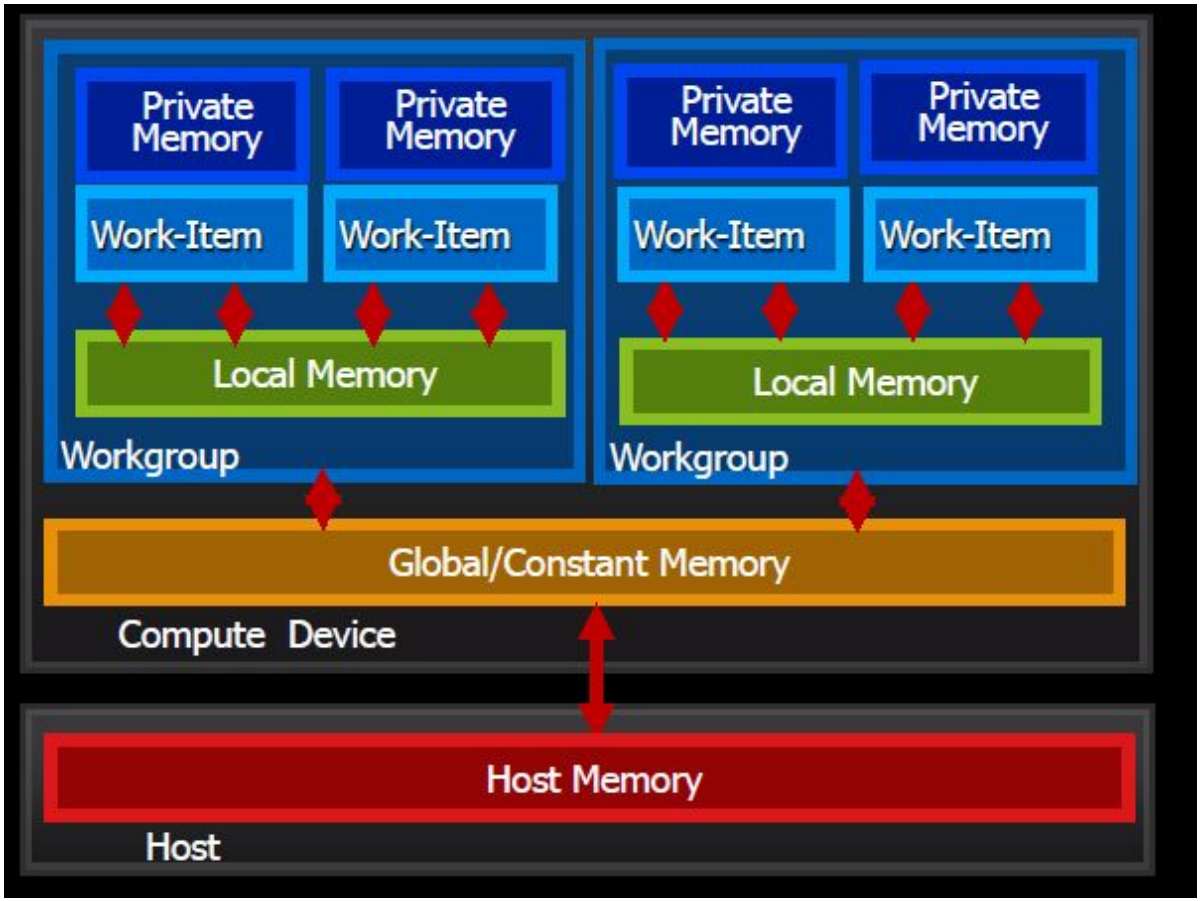
Обычно один элемент Work-Group приходится на один Compute Unit.

Очередь команд OpenCL



Host направляет команды на устройства. Эти команды становятся в очередь аналогичных команд. Можно реализовать очередь с соблюдением порядка и без соблюдения.

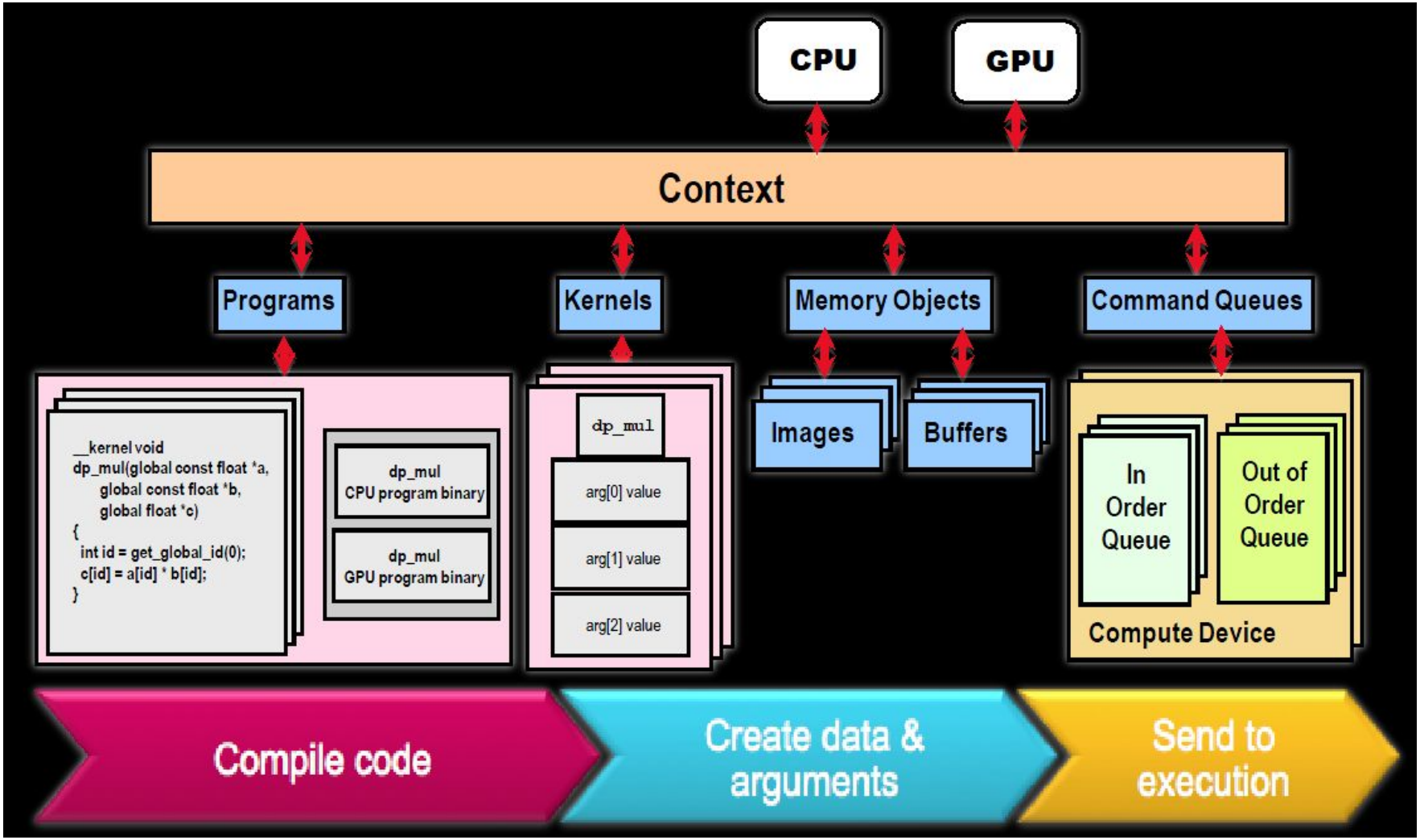
Виды памяти в OpenCL-устройствах



Программист должен явным образом отдавать команды копирования данных между Local, Global и Private Memory.

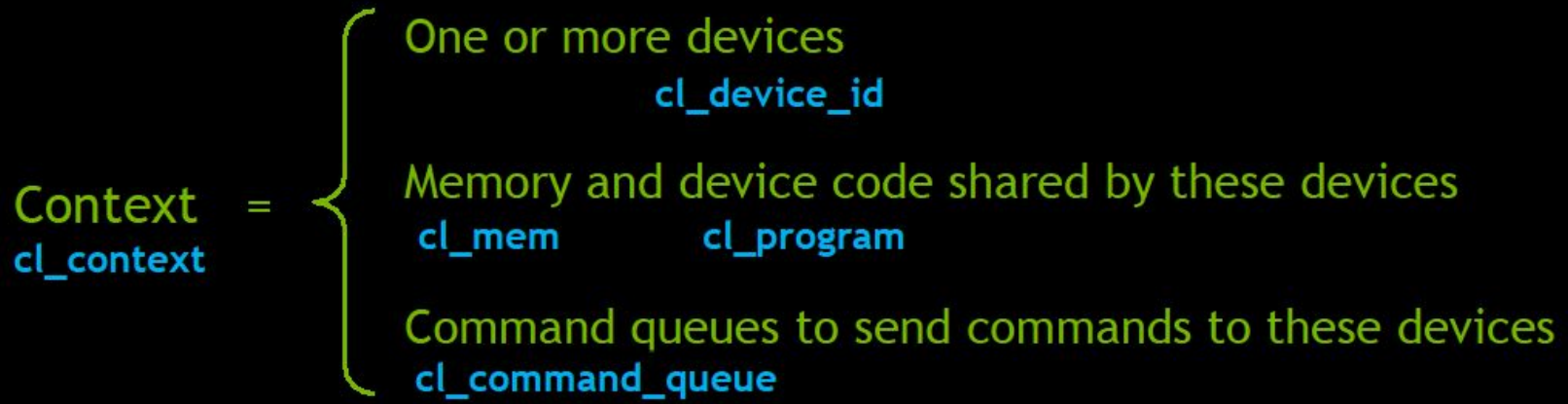
hhttp://habrahabr.ru/post/55461/ - память в CUDA.

Понятие вычислительного контекста в OpenCL

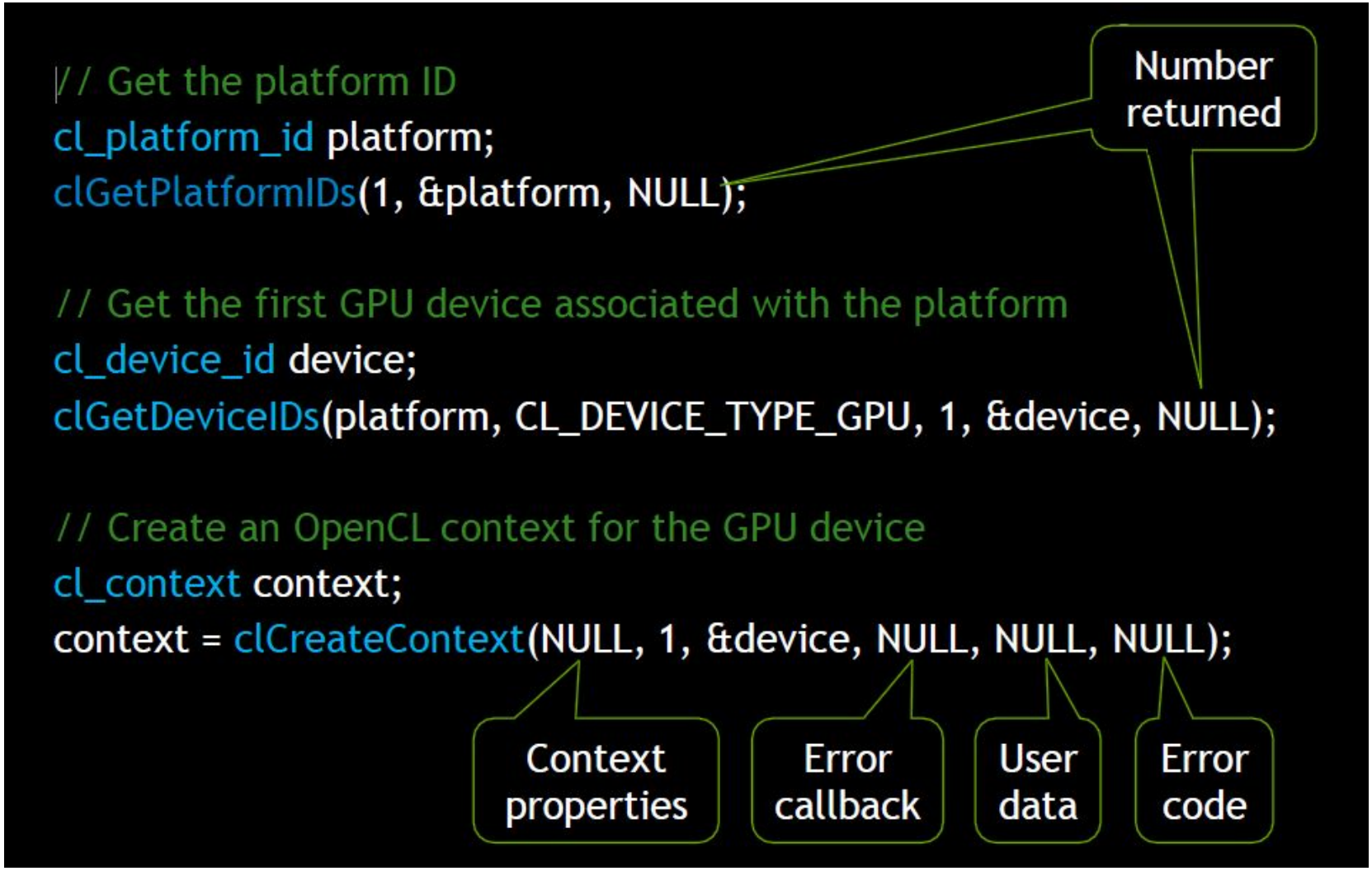


Понятие контекста в OpenCL

- Query platform information
  - clGetPlatformInfo(): profile, version, vendor, extensions
  - clGetDeviceIDs(): list of devices
  - clGetDeviceInfo(): type, capabilities
- Create an OpenCL context for one or more devices



Создание контекста в OpenCL



Принципы работы OpenCL (для 2.2): *оболочка на C*

1. Выбор платформы:

clGetPlatformIDs, clGetPlatformInfo (c. 53, # 4.1)

2. Выбор устройства:

clGetDeviceIDs, clGetDeviceInfo (c. 55, # 4.2)

3. Создание вычислительного контекста:

clCreateContextFromType (c. 77, # 4.4)

4. Создание очереди команд:

clCreateCommandQueueWithProperties (c. 81, # 5.1)

5. Выделение памяти в виде буферов:

clCreateBuffer (c. 86, # 5.2.1)

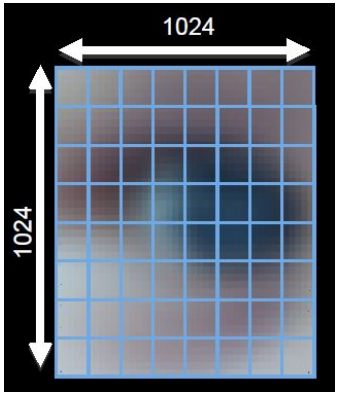
6. Создание объекта «программа»:

clCreateProgramWithSource (c. 146, # 5.8.1)

Простой пример ядра OpenCL

```
void
trad_mul(int n,
         const float *a,
         const float *b,
         float *c)
{
  int i;
  for (i=0; i<n; i++)
    c[i] = a[i] * b[i];
}
```

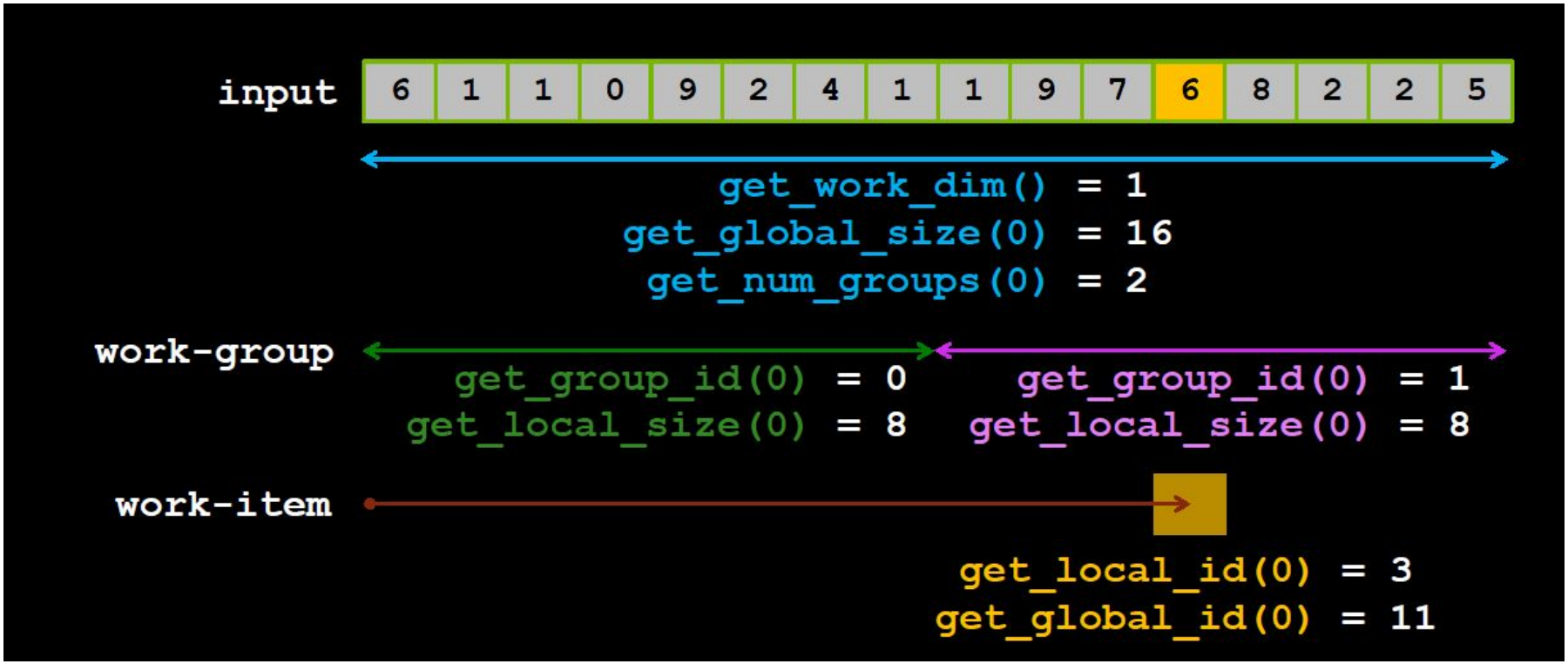
```
__kernel void
dp_mul(__global const float *a,
       __global const float *b,
       __global float *c)

{
  int id = get_global_id(0);

  c[id] = a[id] * b[id];
} // execute over n "work items"
```



n = 1024 – это число work items.
m = 1024/cores – это число work groups.

Работа в рамках одной work group выполняется одновременно всеми work items. 1 WG –> 1 Compute Unit.

Work group и Work item



Нецелое число Work group?

```
__kernel void dp_mul(__global const float *a,
                     __global const float *b,
                     __global float *c,
                     int N)
{
    int id = get_global_id (0);
    if (id < N)
        c[id] = a[id] * b[id];
}
```

Компиляция kernel

```
// Build program object and set up kernel arguments
const char* source = "__kernel void dp_mul(__global const float *a, \n"
                     "                     __global const float *b, \n"
                     "                     __global float *c, \n"
                     "                     int N) \n"
                     "{ \n"
                     "        int id = get_global_id (0); \n"
                     "        if (id < N) \n"
                     "                c[id] = a[id] * b[id]; \n"
                     "} \n";
cl_program program = clCreateProgramWithSource(context, 1, &source, NULL, NULL);
clBuildProgram(program, 0, NULL, NULL, NULL, NULL);
cl_kernel kernel = clCreateKernel(program, “dp_mul", NULL);
clSetKernelArg(kernel, 0, sizeof(cl_mem), (void*)&d_buffer);
clSetKernelArg(kernel, 1, sizeof(int), (void*)&N);
```

Копирование данных с/на device

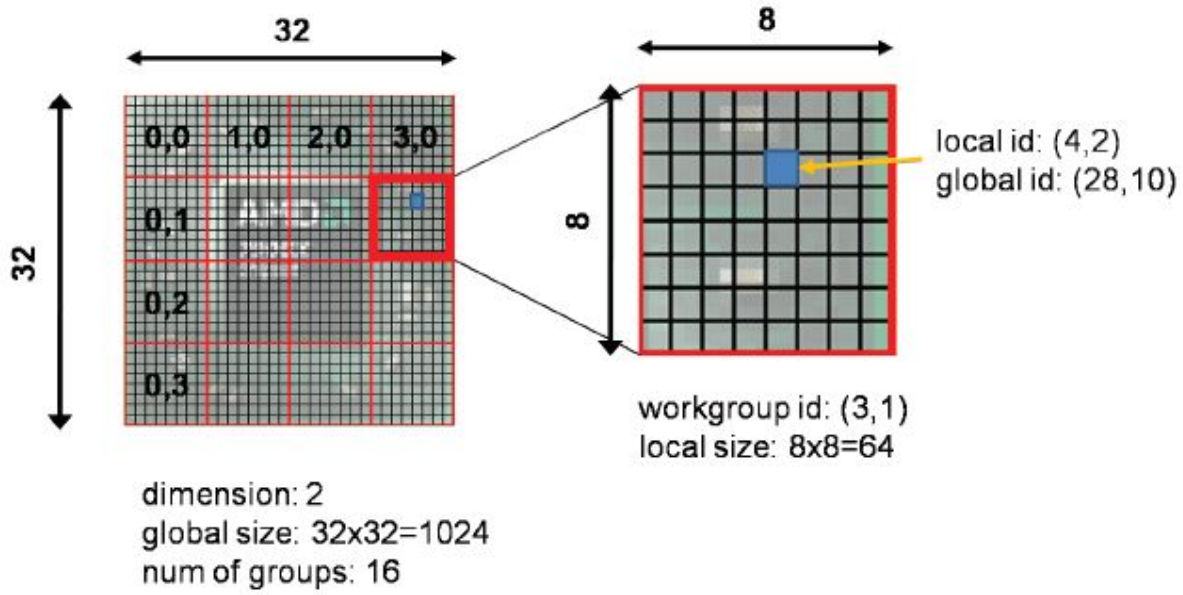
```
// Create buffers on host and device
size_t size = 100000 * sizeof(int);
int* h_buffer = (int*)malloc(size);
cl_mem d_buffer = clCreateBuffer(context, CL_MEM_READ_WRITE, size, NULL, NULL);
…
// Write to buffer object from host memory
clEnqueueWriteBuffer(cmd_queue, d_buffer, CL_FALSE, 0, size, h_buffer, 0, NULL, NULL);
…
// Read from buffer object to host memory
clEnqueueReadBuffer(cmd_queue, d_buffer, CL_TRUE, 0, size, h_buffer, 0, NULL, NULL);
```

Blocking? Offset Event synch

Запуск kernel

```
// Set number of work-items in a work-group
size_t localWorkSize = 256;
int numWorkGroups = (N + localWorkSize - 1) / localWorkSize; // round up
size_t globalWorkSize = numWorkGroups * localWorkSize; // must be evenly divisible by localWorkSize
clEnqueueNDRangeKernel(cmd_queue, kernel, 1, NULL, &globalWorkSize, &localWorkSize, 0, NULL, NULL);
```

NDRange



Запуск kernel

```
// Set number of work-items in a work-group
size_t localWorkSize = 256;
int numWorkGroups = (N + localWorkSize - 1) / localWorkSize; // round up
size_t globalWorkSize = numWorkGroups * localWorkSize; // must be evenly divisible by localWorkSize
clEnqueueNDRangeKernel(cmd_queue, kernel, 1, NULL, &globalWorkSize, &localWorkSize, 0, NULL, NULL);
```

NDRange

```
__kernel void square(const __global float *input0,
                     const __global float *input1,
                     __global float * out)
{
    const int Width = get_global_size(0);
    const size_t xid = get_global_id(0);
    const size_t yid = get_global_id(1);

    const int idx= id*Width + xid;
    out[idx]=input0[idx]+input1[idx];

}
```

Принципы работы OpenCL (для 2.2): *оболочка на C*

7. Компиляция кода:

clBuildProgram (с. 151, # 5.8.4)

CL_BUILD_PROGRAM_FAILURE = код ошибки, тогда вызов clGetProgramBuildInfo с аргументом CL_PROGRAM_BUILD_LOG

8. Создание «ядра» (объект kernel):

clCreateKernel (с. 170, # 5.9.1)

9. Работа с Work-Group:

clGetKernelWorkGroupInfo – с. 238 (# 5.9.4)

Принципы работы OpenCL (для 2.2): *оболочка на C*

10. Выполнение ядра:

clEnqueueNDRangeKernel (с. 187, # 5.10)

11. Ожидание выполнения ядра:

clWaitForEvents (с. 193, # 5.11)

12. Profiling:

clGetEventProfilingInfo (с. 201, # 5.14)

Принципы работы OpenCL: *программа на OpenCL*

__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.

__read_only и __write_only – квалификаторы режима доступа.

Функции Work-Itemов:

get_local_id, get_group_id и т.д.