Parallel Computing

Academic year – 2020/21, spring semester

Computer science

Lecture 8

Lecturer, instructor:

Balakshin Pavel Valerievich

(pvbalakshin@itmo.ru; pvbalakshin@hdu.edu.cn)

Assistant:

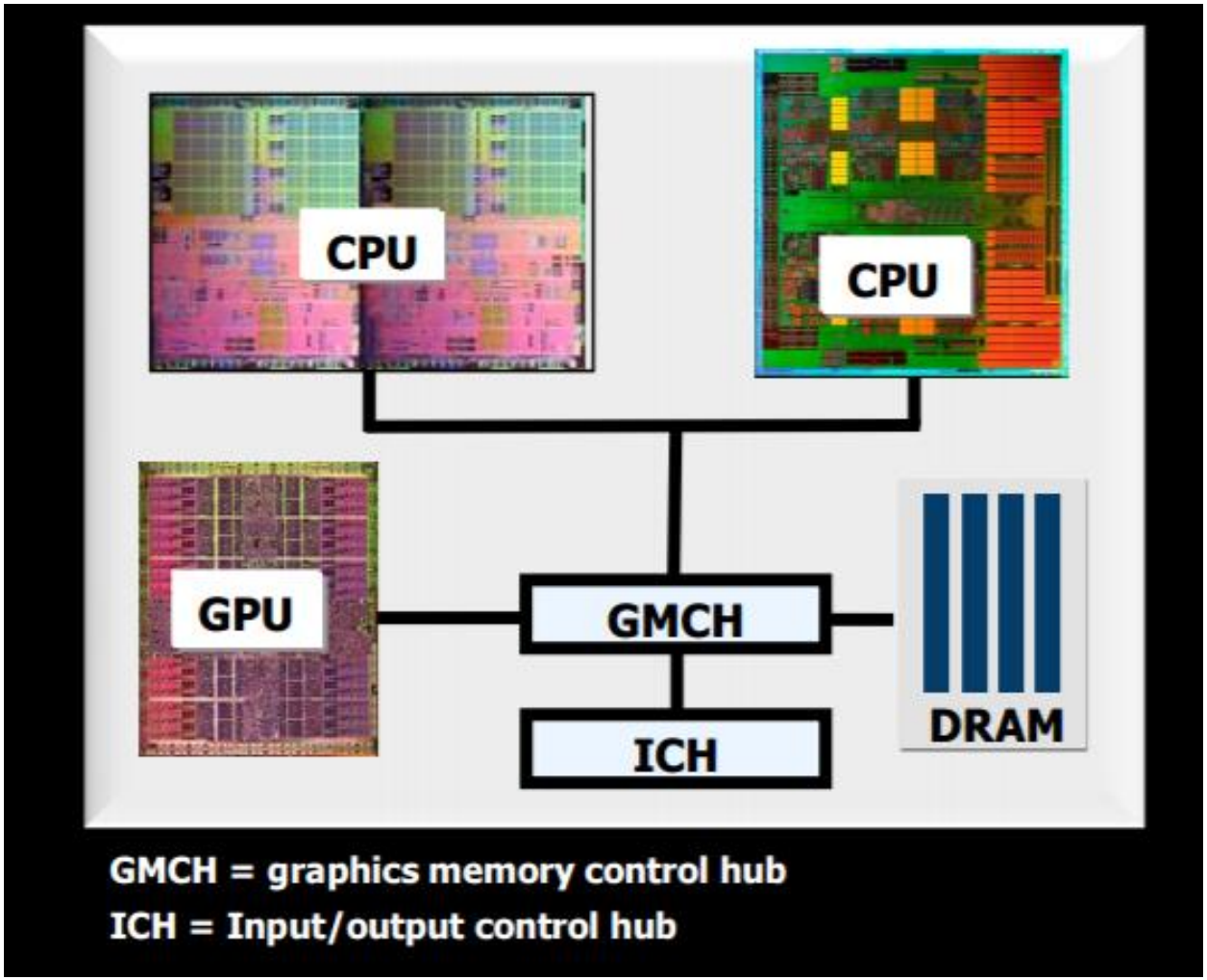
Liang Tingting

(liangtt@hdu.edu.cn)

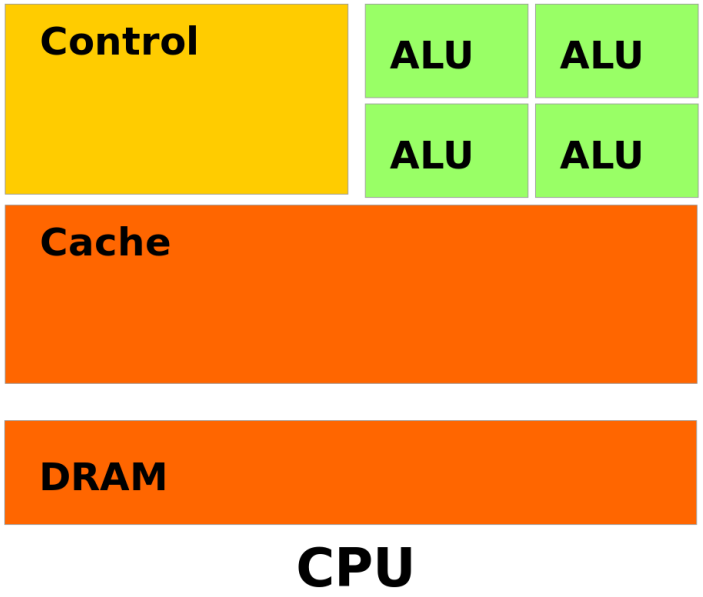
Used materials

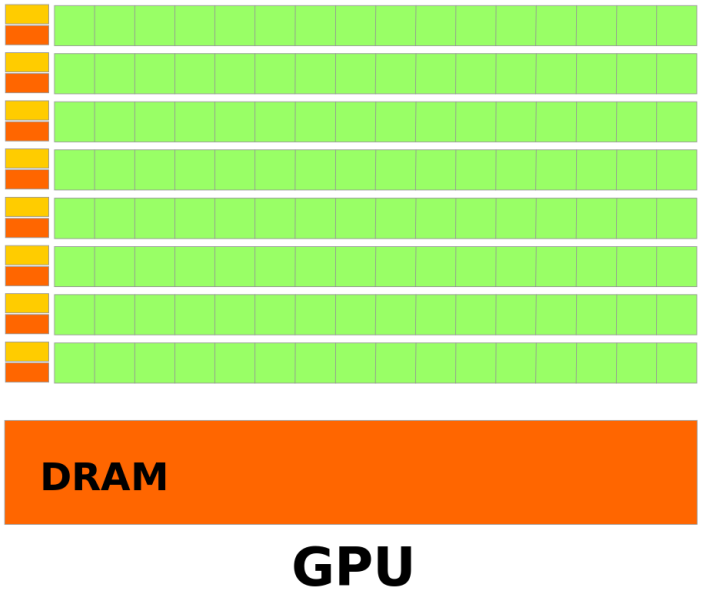
- «Introduction to OpenCL Programming». – AMD, 2010.
- «Introduction to OpenCL». – Nvidia, 2011
- https://www.khronos.org/registry/OpenCL/specs/2.2/pdf/OpenCL_API.pdf
- http://docplayer.ru/37490743-Programmirovanie-na-opencl.html (2011)
- https://www.xilinx.com/html_docs/xilinx2017_4/sdaccel_doc/pet1504034296131.html
- https://www.osp.ru/os/2013/08/13037850
- http://www.hsafoundation.com/
- Intel OpenCL: http://software.intel.com/en-us/articles/intel-opencl-sdk/
- AMD OpenCL: https://community.amd.com/community/devgurus/opencl/content
- NVIDIA CUDA: https://developer.nvidia.com/cuda-zone
- Memory in CUDA: http://habrahabr.ru/post/55461/

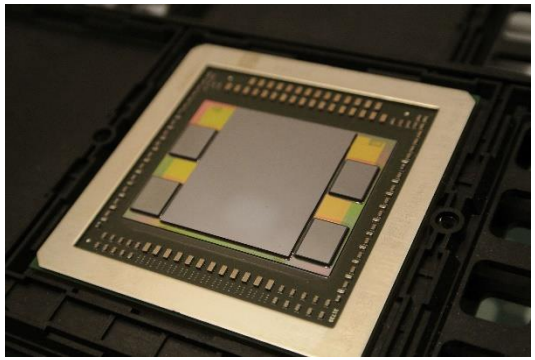
Heterogeneous vs Homogeneous parallel computing

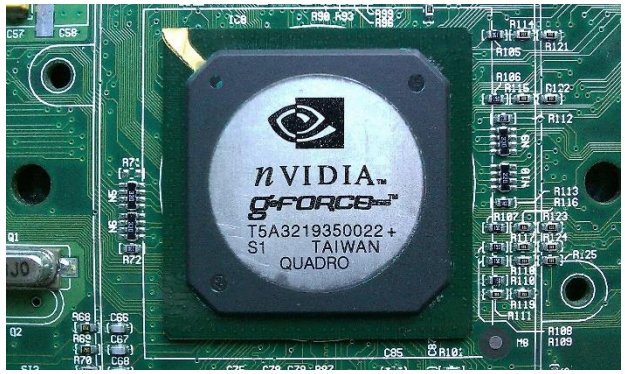


CPU vs GPU


Control
ALU
ALU
ALU
ALU
Cache
DRAM
CPU


DRAM
GPU

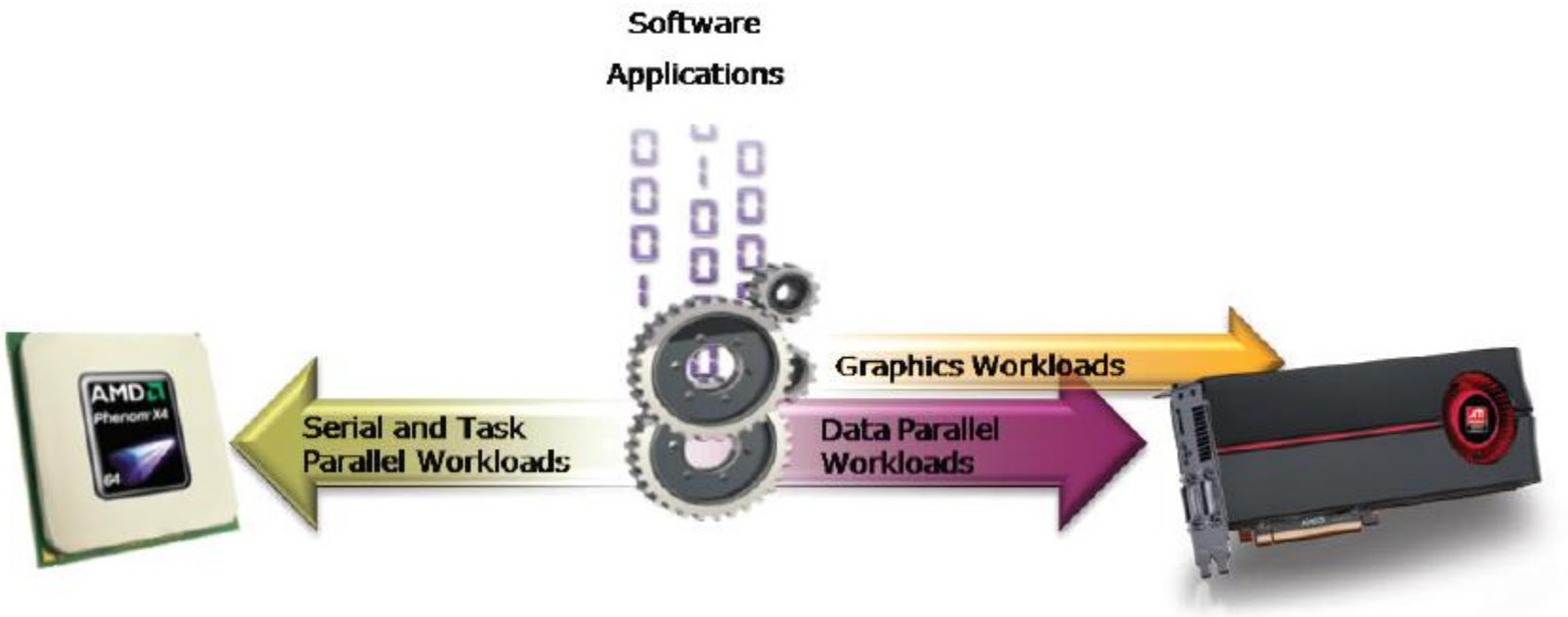



nVIDIA
GeForce
T5A3219350022+
S1 TAIWAN
QUADRO

What OpenCL is?

- OpenCL (Open Computing Language) - a framework for writing computer programs that execute across heterogeneous platforms consisting of central processing units (CPUs), graphics processing units (GPUs), digital signal processors (DSPs), field-programmable gate arrays (FPGAs) and other processors or hardware accelerators.
- The goal of OpenCL is to complement OpenGL and OpenAL, which are open industry standards for 3D computer graphics and sound by taking advantage of the GPU.
- The Khronos Group Consortium, which includes many major companies including Apple, AMD, Intel, nVidia, ARM, Huawei, Sony Computer Entertainment and others.
- The first version of the standard was announced on 9th of December, 2008.
- Current documentation version from 19th of July, 2019: https://www.khronos.org/registry/OpenCL/specs/2.2/pdf/OpenCL_API.pdf.

Typical OpenCL usage model



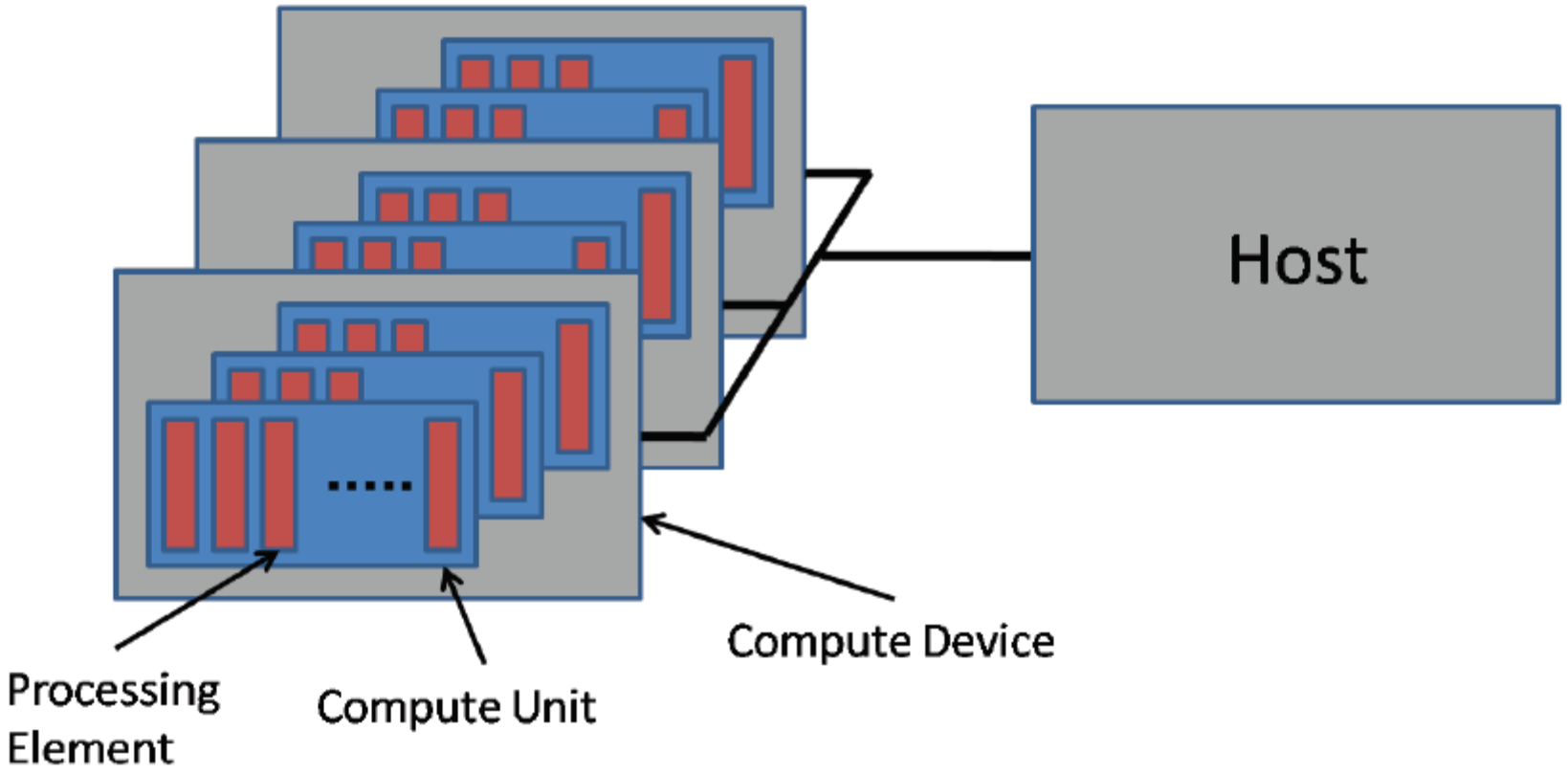
Parallelization by tasks (units/dozens complex productive cores)

Data paralleling (thousands of simple slow cores)

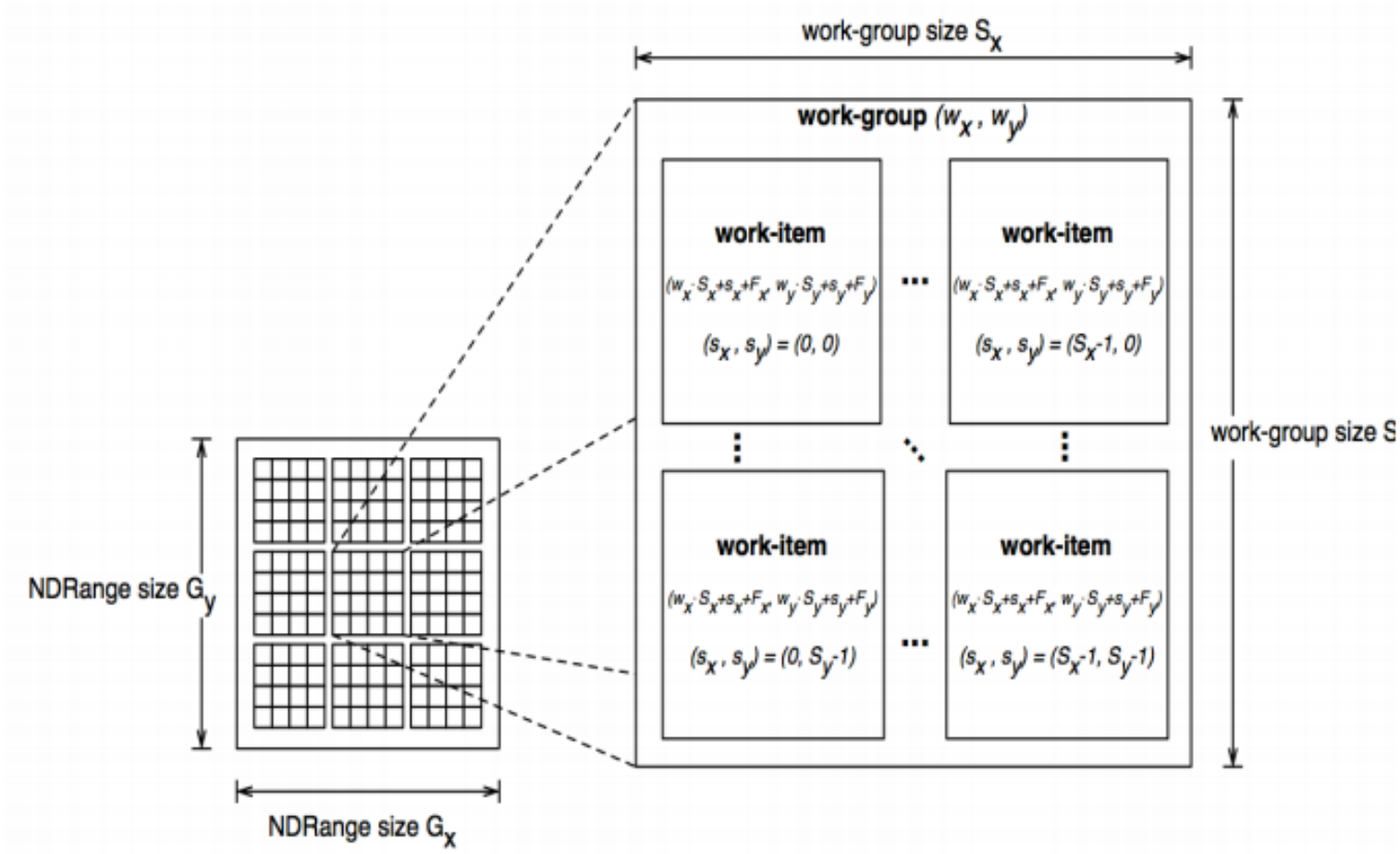
Program language in OpenCL

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

How OpenCL sees the hardware (platform)

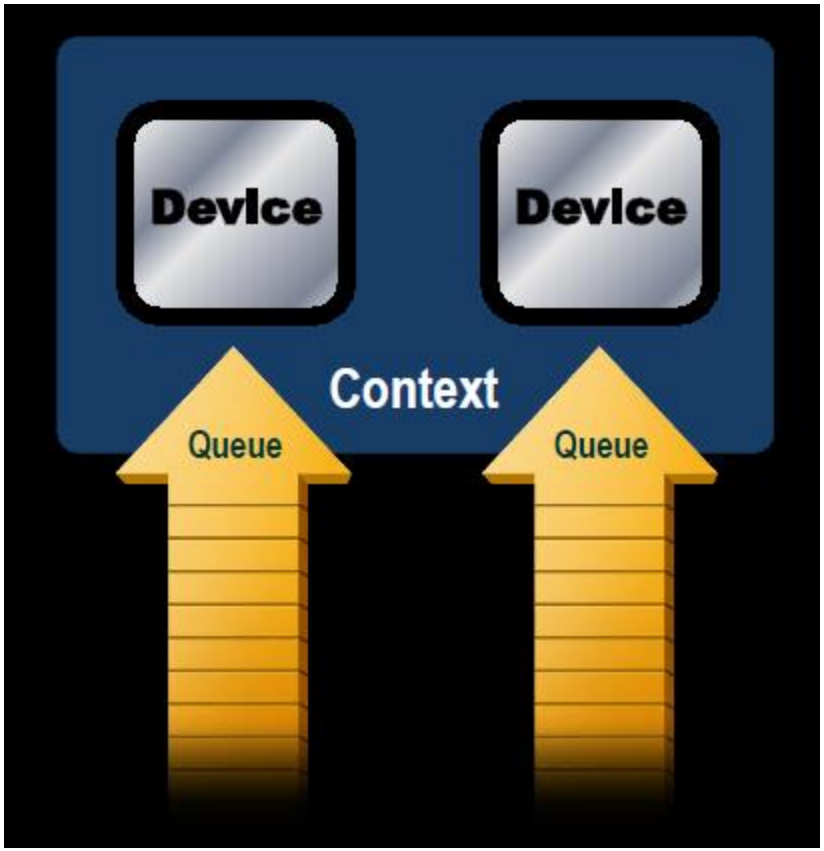


OpenCL working principle



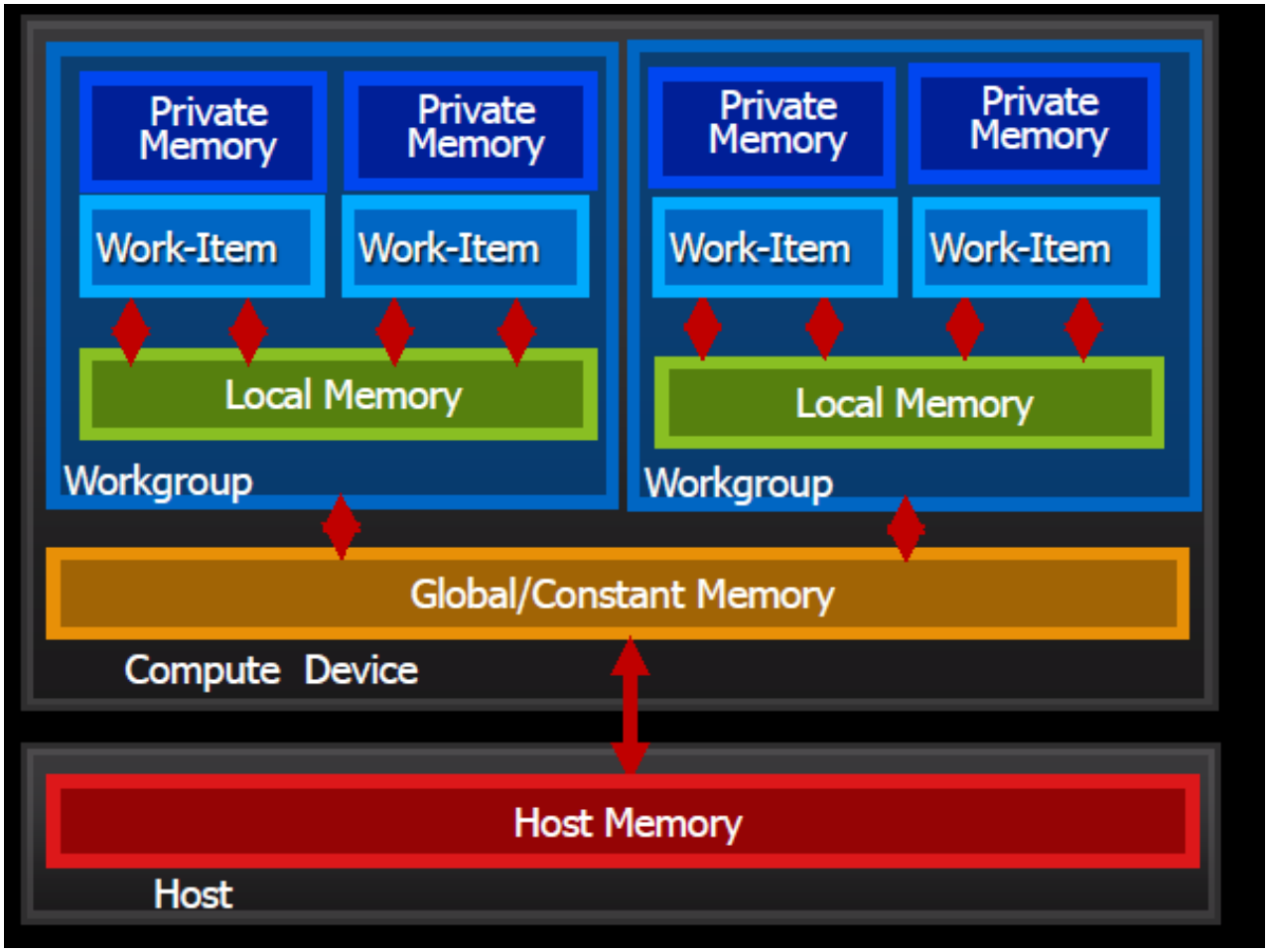
Usually one Work-Group element corresponds to one Compute Unit.

OpenCL command queue



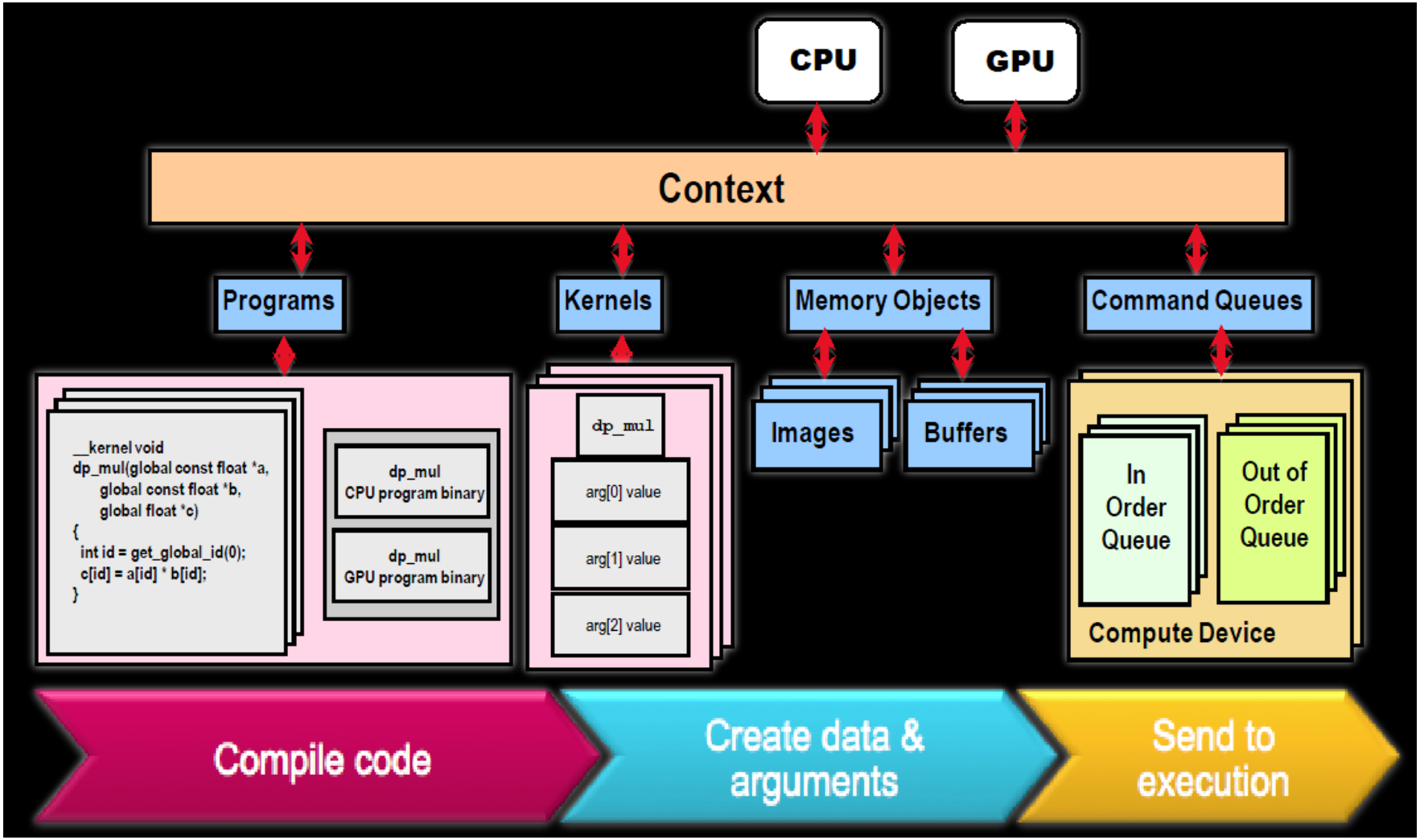
The Host sends commands to the devices.
Those commands become a queue of similar commands.
It is possible to implement a queue in an orderly and non-ordered sequence.

Memory types in OpenCL-devices



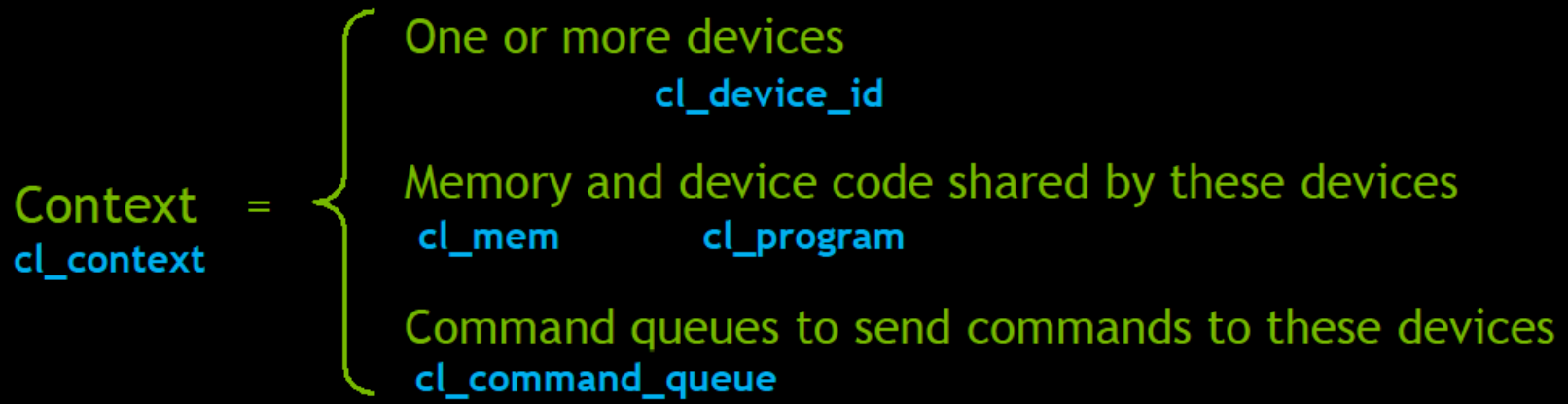
The programmer should explicitly give commands to copy data between Local, Global and Private Memory.

Computational context in OpenCL



Computational context in OpenCL (2)

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



Creation context in OpenCL

```
// Get the platform ID
cl_platform_id platform;
clGetPlatformIDs(1, &platform, NULL);

// Get the first GPU device associated with the platform
cl_device_id device;
clGetDeviceIDs(platform, CL_DEVICE_TYPE_GPU, 1, &device, NULL);

// Create an OpenCL context for the GPU device
cl_context context;
context = clCreateContext(NULL, 1, &device, NULL, NULL, NULL);
```

Number returned

Context properties

Error callback

User data

Error code

OpenCL work principles (for 2.2): *context on C*

1. Platform choice:

clGetPlatformIDs, clGetPlatformInfo (p. 51, # 4.1)

2. Device choice:

clGetDeviceIDs, clGetDeviceInfo (p. 53, # 4.2)

3. Computational context creation:

clCreateContextFromType (p. 76, # 4.4)

4. Commands queue creation:

clCreateCommandQueueWithProperties (p. 83, # 5.1)

5. Memory allocation using buffers:

clCreateBuffer (p. 89, # 5.2.1)

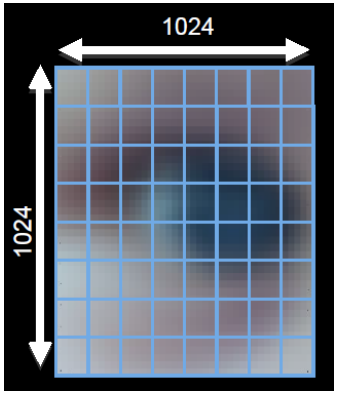
6. Creation of “program” object:

clCreateProgramWithSource (p. 168, # 5.8.1)

Simple kernel example

```
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];
}
```

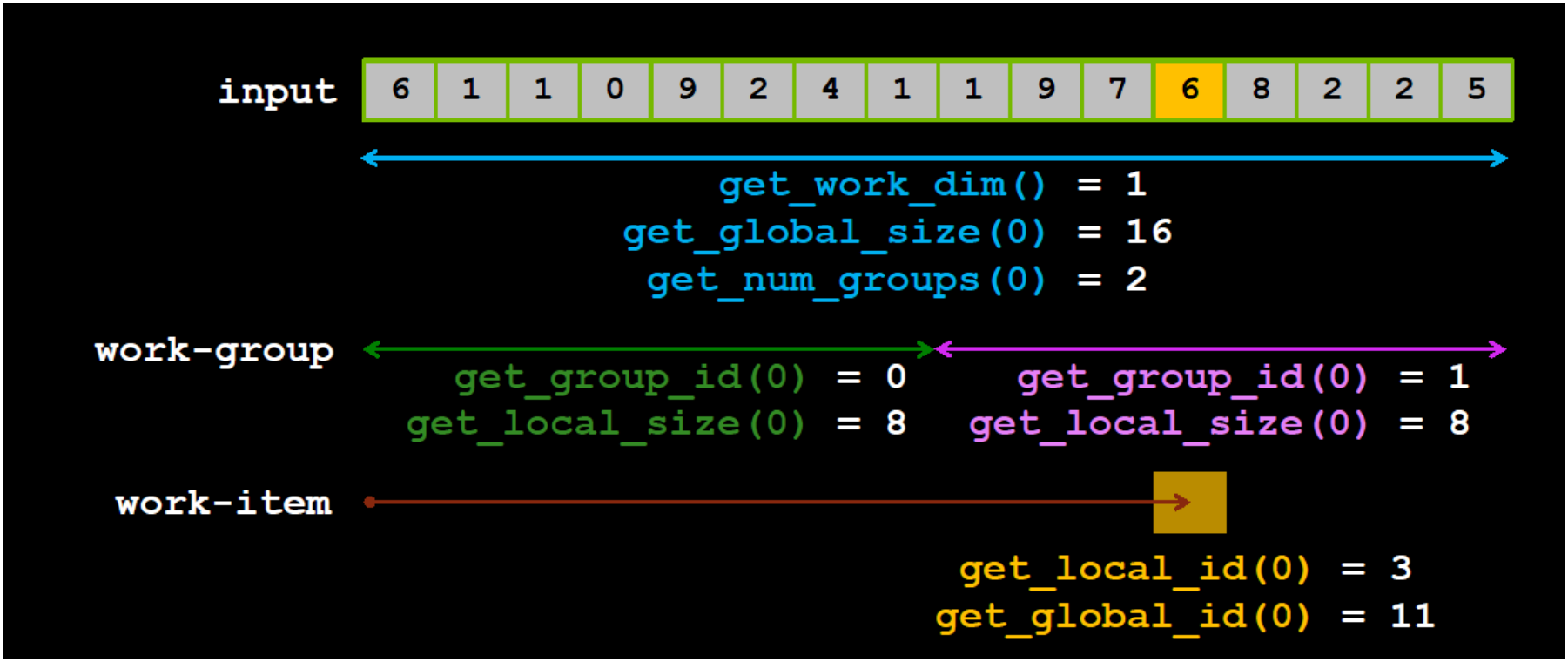
```
__kernel void
dp_mul(__global const float *a,
       __global const float *b,
       __global float *c)

{
  int id = get_global_id(0);

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



n = 1024 is the number of work items.
m = 1024/cores is the number of work groups.

Work within one work group is performed simultaneously by all the work items. 1 WG –> 1 Compute Unit.

Work group and work item



Partial work group items

```
__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];
}
```

OpenCL work principles (for 2.2): *context on C* (2)

7. Compiling of kernel:

clBuildProgram (p. 175, # 5.8.4)

CL_BUILD_PROGRAM_FAILURE = error code, them invocation of clGetProgramBuildInfo with CL_PROGRAM_BUILD_LOG parameter

8. Creation of object kernel:

clCreateKernel (p. 196, # 5.9.1)

9. Work with Work-Group:

clGetKernelWorkGroupInfo (p. 208, # 5.9.4)

Compiling of kernel (2)

```
// 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);
```

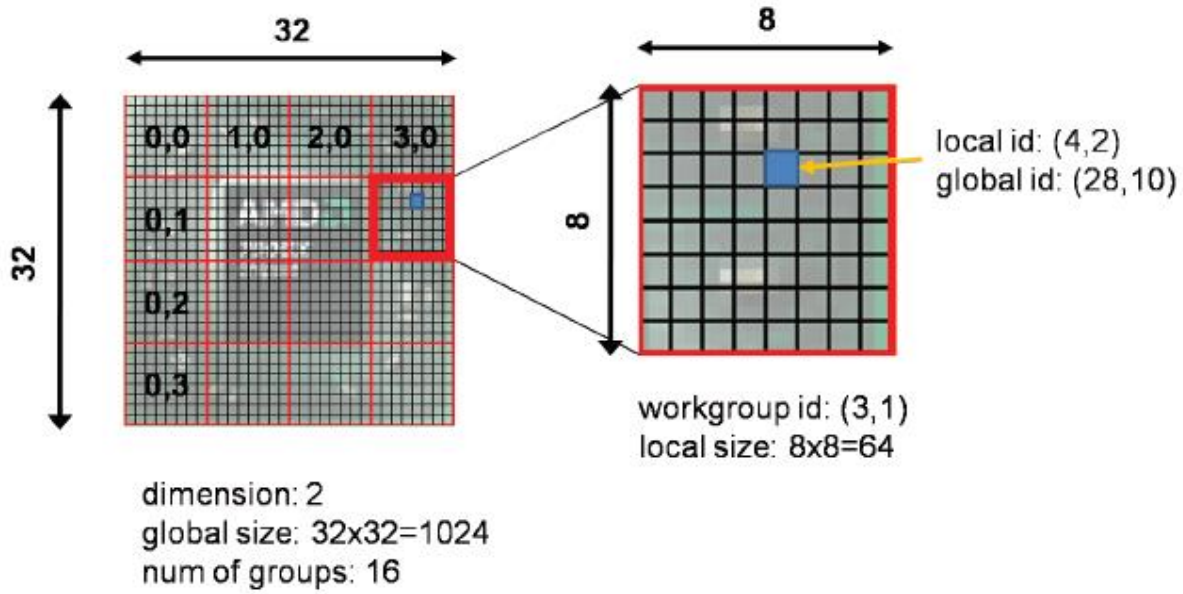
Copy data from/to device

```
// 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);
```

Execution of 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



Execution of kernel (2)

```
// 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];

}
```

OpenCL work principles (for 2.2): *context on C* (3)

10. Execution of kernel:

clEnqueueNDRangeKernel (p. 217, # 5.10)

11. Waiting for execution of kernel:

clWaitForEvents (p. 222, # 5.11)

12. Profiling:

clGetEventProfilingInfo (p. 235, # 5.14)

OpenCL work principles (for 2.2): *program on OpenCL*

__global or global – global memory data.

__constant or constant – constant memory data.

__local or local – local memory data.

__private or private – private memory data.

__read_only and __write_only – data access quilifiers.

Work-Items functions:

get_local_id, get_group_id, etc.