OpenCL

Build and run OpenCL applications

GPU Programming

http://cuda.nik.uni-obuda.hu

Szénási Sándor
szenasi.sandor@nik.uni-obuda.hu

GPU Education Center of Óbuda University

NVIDIA GPU EDUCATION CENTER

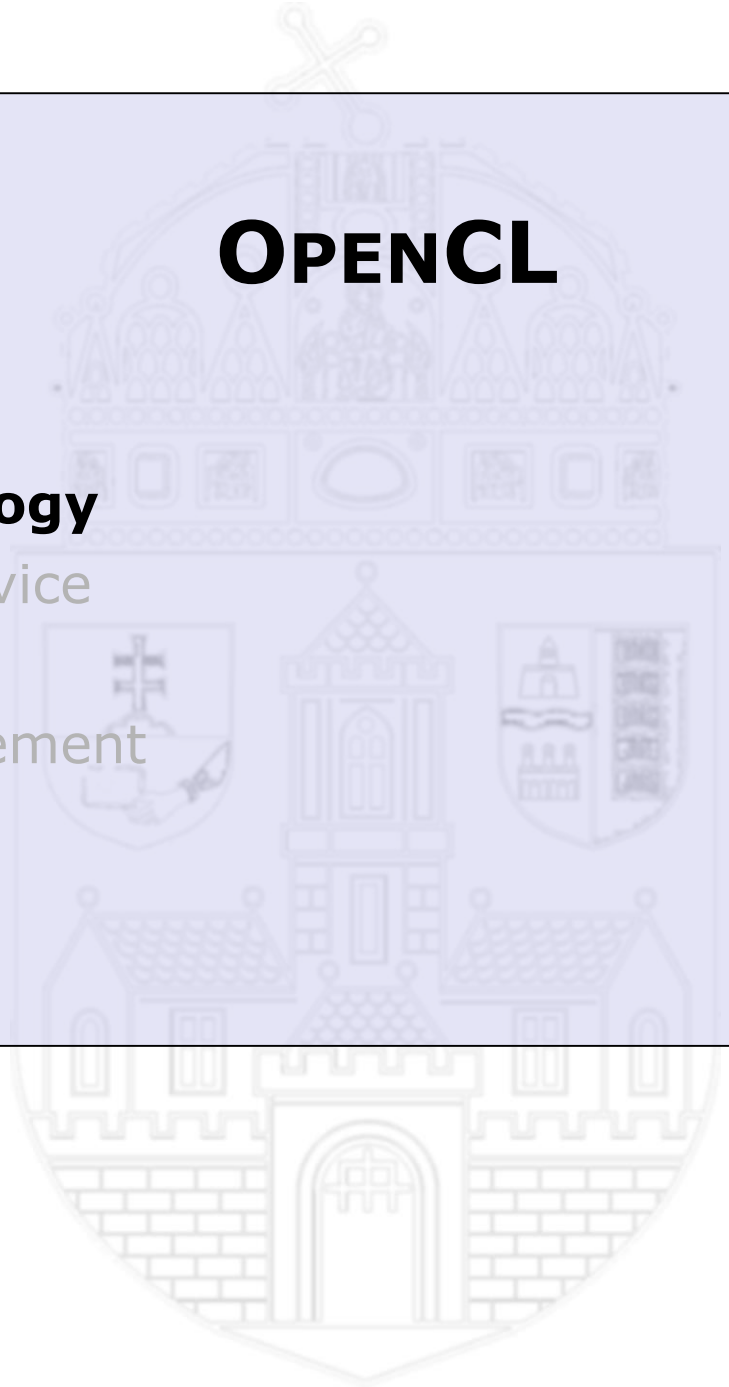
OpenCL

Basic terminology

Platform and device

Launch kernel

Memory management



OpenCL framework

What is OpenCL

- An open royalty-free standard for general purpose parallel programming across CPUs, GPUs and other processors, giving software developers portable and efficient access to the power of these heterogeneous processing platforms
- Main models
  - platform model
  - memory model
  - execution model
  - programming model

Developers

- Specification
  - Khronos Group
  - AMD, Apple, Intel, Nvidia, etc.
- Implementations
  - AMD
  - Apple
  - Nvida
  - Intel
  - etc.

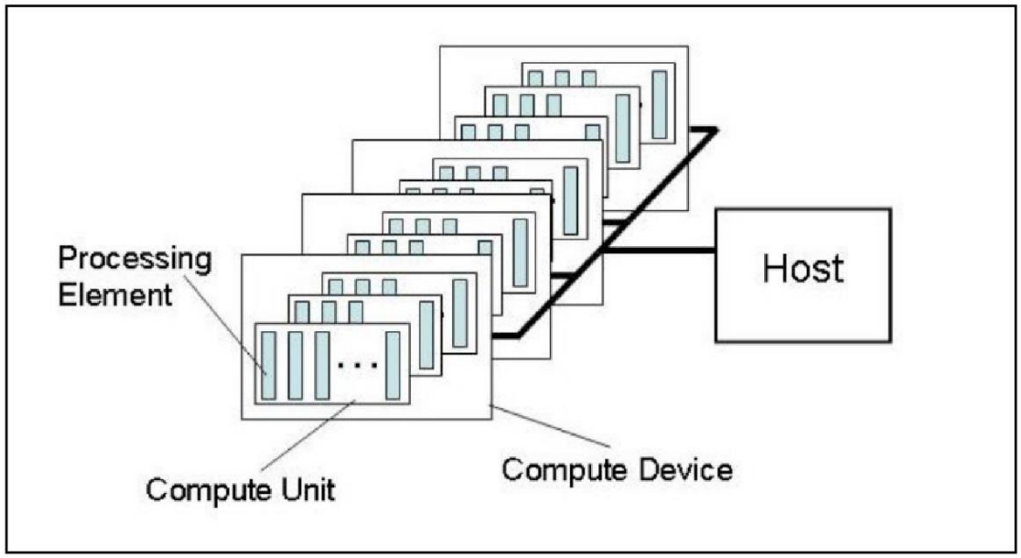
Platform model

Elements

- One host
- More OpenCL devices

Inside one device

- One or more compute units
- Inside compute units, there are one or more processing elements



[i21]

Program compilation

- Online compiler – during host program execution
- Offline - outside of host program control

Platform model

- An abstraction describing how OpenCL views the hardware

Execution model

Contexts

- Kernel executes within contexts managed by the host
- A context defines the followings
  - devices: one more devices
  - kernel objects: OpenCL functions that run on OpenCL devices
  - program objects: the implementation of the kernel (source and binary)
  - memory objects: variables visible to the host and the OpenCL devices

Command-queue

- Each command-queue is associated with one device
- Command types
  - kernel-enqueuer commands
  - memory commands
  - synchronization commands
- Command states:
  queued → submitted → ready → running → ended → complete
- Commands communicate their status through event objects
- Command execution can be
  - blocking or non-blocking
  - in-order or out-of-order

NDRange

Kernel execution

- For kernel execution commands, an index space is defined
- Each executing kernel functions is called a work-item
- Work-items associated in groups called work-groups
- Work-groups are further divided into sub-groups (implementation-defined)
- Work-items have a local ID and a global ID
- Work-groups are assigned unique identifiers

NDRange

- The index space supported by OpenCL is called and NDRange
- NDRange is a 1,2 or 3 dimensional index space
- NDRange is decomposed into work-groups
- An NDRange is defined by three integers arrays of length 1,2 or 3
  - the extent of the index space (in each dimension)
  - an offset index indicating the initial value (in each dimension)
  - the size of a work-group (in each dimension)

Domains of synchronization

Work-group synchronization

- Constraints on the order of execution for work-items in a single work-group
- Available collective operations
  - barrier
  - reduction
  - broadcast
  - prefix sum
  - evaluation of a predicate

Sub-group synchronization

- Constraints on the order of execution for work-items in a single sub-group

Command synchronization

- Launching commands
- Ending a command
- Completion of a command
- Blocking a command
- Command-queue barrier
- Finish

Memory regions

Host memory

- Directly available to the host
- Memory objects can move between the host and devices

Device memory

- Global memory
  - r/w access to all work-items
  - may be cached
- Constant memory
  - host can allocate and initialize
  - constant during kernel execution
- Local memory
  - memory region local to a work-group
  - for variables shared between work-items of the same group
- Private memory
  - memory region private to a work-item
  - variables here are not visible for other work-items

Generic address space

- Global – local – private memory regions

CUDA – OpenCL terminology

Main terms

CUDA C	OpenCL
Thread	Work-item
Thread block	Work-group
Global memory	Global memory
Constant memory	Constant memory
Shared memory	Local memory
Local memory	Private memory

Function and variable qualifiers

CUDA C	OpenCL
__global__	__kernel
__device__	
__constant__	__constant
__device__	__global
__shared__	__local

CUDA – OpenCL terminology (kernel code)

Indexing in kernel code

CUDA C	OpenCL
gridDim	get_num_groups()
blockDim	get_local_size()
blockIdx	get_group_id()
threadIdx	get_local_id()
blockIdx * blockDim + threadIdx	get_global_id()
gridDim * blockDim	get_global_size()

Synchronization in kernel code

CUDA C	OpenCL
__syncthreads()	barrier(CLK_GLOBAL_MEM_FENCE | CLK_LOCAL_MEM_FENCE)
__threadfence()	No direct equivalent.
__threadfence_block()	mem_fence(CLK_GLOBAL_MEM_FENCE | CLK_LOCAL_MEM_FENCE)
	read_mem_fence()
	write_mem_fence()

OpenCL

Basic terminology

Platform and device

Launch kernel

Memory management

Get platforms

OpenCL platform

- Platform is a specific OpenCL implementation
- There are several already existing platforms, for example
  - AMD APP
  - NVIDIA
  - Intel OpenCL
- There can be multiple platforms available in one host

Get list of platforms

- Function name: clGetPlatformIDs
- Parameters
  - cl_uint num_entries - Maximum number of entries
  - cl_platform_id *platforms - Returns a list of platforms
  - cl_uint *num_platforms - Return the number of platforms
- Possible return values
  - CL_SUCCESS, CL_INVALID_VALUE, CL_OUT_OF_HOST_MEMORY
- The list of platforms available can be obtained using this function.

Get information about one platform

Query platform information

- Function name: clGetPlatformInfo
- Parameters
  - cl_platform_id platform - platform identifier
  - cl_platform_info param_name - platform information being queried
  - size_t param_value_size - size of memory pointed by value
  - void *param_value - pointer to memory location for data
  - size_t *param_value_size_ret - actual size of data being queried
- Possible return values
  - CL_INVALID_PLATFORM, CL_INVALID_VALUE, CL_OUT_OF_HOST_MEMORY

Platform infos

- CL_PLATFORM_PROFILE - char[]
- CL_PLATFORM_VERSION - char[]
- CL_PLATFORM_NAME - char[]
- CL_PLATFORM_VENDOR - char[]
- CL_PLATFORM_EXTENSIONS - char[]
- CL_PLATFORM_HOST_TIMER_RESOLUTION - cl_ulong

Get devices

OpenCL device

- A device is a collection of compute units
- OpenCL devices typically correspond to a GPU, a multi-core CPU, and other processors such as DSPs and the Cell/B.E. processor
- There can be multiple devices of one platform

Get list of available devices

- Function name: clGetDeviceIDs
- Parameters
  - cl_platform_id platform - current platform
  - cl_device_type device_type - filtering device types
  - cl_uint num_entries - maximum number of entries
  - cl_device_id *devices - pointer for result data
  - cl_uint *num_devices - number of queried devices
- Possible return values
  - CL_SUCCESS or some error code
- Device type can be one of the followings
  - CL_DEVICE_TYPE_CPU, CL_DEVICE_TYPE_GPU, CL_DEVICE_TYPE_ACCELERATOR, CL_DEVICE_TYPE_CUSTOM, CL_DEVICE_TYPE_DEFAULT, CL_DEVICE_TYPE_ALL

Get information about one device

Query device information

- Function name: clGetDeviceInfo
- Parameters
  - cl_device_id device - device identifier
  - cl_device_info param_name - device information being queried
  - size_t param_value_size - size of memory pointed by value
  - void *param_value, - pointer to memory location for data
  - size_t *param_value_size_ret - actual size of data being queried
- Possible return values
  - CL_SUCCESS, CL_INVALID_DEVICE, CL_INVALID_VALUE, CL_OUT_OF_RESOURCES, CL_OUT_OF_HOST_MEMORY

Device info parameters

- CL_DEVICE_TYPE - cl_device_type
- CL_DEVICE_VENDOR_ID - cl_uint
- CL_DEVICE_MAX_COMPUTE_UNITS - cl_uint
- CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS - cl_uint
- CL_DEVICE_MAX_WORK_GROUP_SIZE - cl_uint
- CL_DEVICE_MAX_CLOCK_FREQUENCY - cl_uint
- CL_DEVICE_MAX_MEM_ALLOC_SIZE - cl_ulong

Get information about one device (info parameters)

Device info parameters

- CL_DEVICE_IL_VERSION - char[]
- CL_DEVICE_GLOBAL_MEM_SIZE - cl_ulong
- CL_DEVICE_MAX_CONSTANT_BUFFER_SIZE - cl_ulong
- CL_DEVICE_LOCAL_MEM_SIZE - cl_ulong
- CL_DEVICE_AVAILABLE - cl_bool
- CL_DEVICE_COMPILER_AVAILABLE - cl_bool
- CL_DEVICE_LINKER_AVAILABLE - cl_bool
- CL_DEVICE_PLATFORM - cl_platform_id
- CL_DEVICE_NAME - char[]
- CL_DEVICE_VENDOR - char[]
- CL_DRIVER_VERSION - char[]
- CL_DEVICE_PROFILE - char[]
- CL_DEVICE_VERSION - char[]
- CL_DEVICE_OPENCL_C_VERSION - char[]
- etc.

Compile OpenCL application

Intel environment

- Download the appropriate platform driver
  - OpenCL™ Runtime Driver for Intel® CPU and Intel® Xeon Phi™ coprocessors for Windows* (64-bit)
- Install the driver

NVIDIA environment

- Download the appropriate platform driver
  - Latest CUDA SDK
- Install the driver

Visual Studio settings

- Win32 console application
- Additional include directories:
  c:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v6.5\include
- Linker/Input/Additional dependencies
  c:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v6.5\lib\Win32\OpenCL.lib

OpenCL

Basic terminology

Platform and device

Launch kernel

Memory management

Create context

OpenCL context

- Contexts are used by the OpenCL runtime for managing objects such as command-queues, memory, program and kernel objects and for executing kernels on one or more devices specified in the context

Create a new context

- Function name: clCreateContext
- Parameters
  - const cl_context_properties *properties - list of properties
  - cl_uint num_devices - number of devices
  - const cl_device_id *devices - list of devices
  - callback function - notify function
  - void *user_data - passed to notify function
  - cl_int *errcode_ret - return error code
- errcode_ret values
  - CL_SUCCESS, CL_INVALID_PLATFORM, CL_INVALID_PROPERTY, CL_INVALID_VALUE, CL_INVALID_VALUE, CL_INVALID_VALUE, CL_INVALID_DEVICE, CL_DEVICE_NOT_AVAILABLE, CL_OUT_OF_RESOURCES, CL_OUT_OF_HOST_MEMORY
- Return value is a created context (type of cl_context)

Manage context

Retain and release context

- Retain
  - Function: cl_int clRetainContext (cl_context context)
  - Increments the context reference count
- Release
  - Function: cl_int clReleaseContext (cl_context context)
  - Decrements the context reference count
  - After the context reference count becomes zero and all the objects attached to context (such as memory objects, command-queues) are released, the context is deleted

Get context info

- Function name: clGetContextInfo
- Parameters
  - cl_context context - context
  - cl_context_info param_name - parameter name
  - size_t param_value_size - maximum value size
  - void *param_value - pointer to memory for data
  - size_t *param_value_size_ret - return data size
- Return value: error code

Create command queue

OpenCL command queue

- OpenCL operations are performed using a command-queue
- Having multiple command-queues allows applications to queue multiple independent commands without requiring synchronization

Create a new command queue

- Function name: clCreateCommandQueueWithProperties
- Parameters
  - cl_context context - valid OpenCL context
  - cl_device_id device - a device from this context
  - const cl_queue_properties *properties - list of properties
  - cl_int *errcode_ret - result error code
- errcode_ret values
  - CL_SUCCESS, CL_INVALID_CONTEXT, CL_INVALID_DEVICE, CL_INVALID_VALUE, CL_INVALID_QUEUE_PROPERTIES, CL_OUT_OF_RESOURCES, CL_OUT_OF_HOST_MEMORY
- Return value is a created command queue (type of cl_command_queue)

Manage command queue

Retain and release command queue

- Retain
  - Function: cl_int clRetainCommandQueue (cl_command_queue cq)
  - Increments the command queue reference count
- Release
  - Function: cl_int clReleaseCommandQueue (cl_command_queue cq)
  - Decrements the command queue reference count
  - After the command_queue reference count becomes zero and all commands queued to command_queue have finished (eg. kernel-instances, memory object updates etc.), the command-queue is deleted

Get command queue info

- Function name: clGetCommandQueueInfo
- Parameters
  - cl_command_queue command_queue - command queue
  - cl_command_queue_info param_name - parameter name
  - size_t param_value_size - maximum value size
  - void *param_value - pointer to memory region
  - size_t *param_value_size_ret - return size
- Return value: error code

Not discussed advanced topics

Sub-buffers

- It is possible to create sub buffers based on already existing buffers

Fill operation

- To fill a buffer using a given pattern

Advanced copy operations

- Copy a 2D or 3D rectangular region from host/device to host/device

Mapping buffer objects

- Map a region of the buffer into the host adress space
- If a memory object is currently mapped for writing, the application must ensure that the memory object is unmapped before any enqueued kernels or commands that read from or write to this memory object
- Same for mapped for reading/kernel writing

Image objects

- Create and handle image objects

Pipes

- A pipe is a memory object that stores data organized as a FIFO

Create program

OpenCL program object

- An OpenCL program consists of a set of kernels that are identified as functions declared with the __kernel qualifier in the program source

Create a new program object using source

- Function name: clCreateProgramWithSource
- Parameters
  - cl_context context - a valid context associated with the program
  - cl_uint count - number of sources
  - const char **strings - array of pointers to sources
  - const size_t *lengths - array of source lengths
  - cl_int *errcode_ret - return error code
- Possible return values
  - CL_SUCCESS, CL_INVALID_CONTEXT, CL_INVALID_VALUE, CL_OUT_OF_RESOURCES, CL_OUT_OF_HOST_MEMORY

Create a new program object using IL code

- Function name: clCreateProgramWithIL
- We do not discuss

Manage program object

Retain and release program object

- Retain
  - Function: cl_int clRetainProgram (cl_program program)
  - Increments the program object reference count
- Release
  - Function: cl_int clReleaseProgram (cl_program program)
  - Decrements the program object reference count
  - The program object is deleted after all kernel objects associated with program have been deleted and the program reference count becomes zero

Get information about a program

- Function name: clGetProgramInfo
- Parameters
  - cl_program program - the program
  - cl_program_info param_name - parameter name
  - size_t param_value_size - maximum size of data buffer
  - void *param_value - pointer to data buffer
  - size_t *param_value_size_ret - return size
- Possible return values
  - CL_SUCCESS or some error code

Build program

Building program executables

- Function name: clBuildProgram
- Parameters
  - cl_program program - the program object
  - cl_uint num_devices - number of devices associated with program
  - const cl_device_id *device_list - list of devices
  - const char *options - null-terminated character with options
  - callback function - notify function
  - void *user_data - user data used by the notify function
- Possible return values
  - CL_SUCCESS or some error code
- OpenCL specification contains the list of valid compiler options

Separate compilation and linking of programs

- Compile a program object
  - function name: clCompileProgram
- Link a program
  - function name: clLinkProgram

Create kernel

OpenCL kernel object

- A kernel is a function declared in a program
- A kernel is identified by the __kernel qualifier applied to any function

Create a new program object using source

- Function name: clCreateKernel
- Parameters
  - cl_program program - a successfully built program
  - const char *kernel_name - name of the kernel
  - cl_int *errcode_ret - return error code
- errcode_ret values
  - CL_SUCCESS, CL_INVALID_PROGRAM, CL_INVALID_PROGRAM_EXECUTABLE, CL_INVALID_KERNEL_NAME, CL_INVALID_KERNEL_DEFINITION, CL_INVALID_VALUE, CL_OUT_OF_RESOURCES, CL_OUT_OF_HOST_MEMORY
- Return value is a created kernel object (type of cl_kernel)
- Possible return values
  - CL_SUCCESS, CL_INVALID_CONTEXT, CL_INVALID_VALUE, CL_OUT_OF_RESOURCES, CL_OUT_OF_HOST_MEMORY

Manage kernel object

Building multiple kernels

- Function name: clCreateKernelsInProgram
- Parameters
  - cl_program program - program object
  - cl_uint num_kernels - maximum number of kernels
  - cl_kernel *kernels - pointer to memory space for kernels
  - cl_uint *num_kernels_ret - number of kernels
- Possible return values
  - CL_SUCCESS or some error code

Retain and release kernel object

- Retain
  - Function: cl_int clRetainKernel (cl_kernel kernel)
  - Increments the kernel object reference count
- Release
  - Function: cl_int clReleaseKernel (cl_kernel kernel)
  - Decrements the kernel object reference count
  - The kernel object is deleted once the number of instances that are retained to kernel become zero and the kernel object is no longer needed by any enqueued commands that use kernel

Setting kernel arguments

Set kernel argument

- Function name: clSetKernelArg
- Parameters
  - cl_kernel kernel - kernel object
  - cl_uint arg_index - argument index
  - size_t arg_size - argument size
  - const void *arg_value - pointer to argument value
- Possible return values
  - CL_SUCCESS or some error code

About parameters

- The index of the first parameter is 0

Launch a kernel

Set kernel argument

- Function name: clEnqueueNDRangeKernel
- Parameters
  - cl_command_queue command_queue - valid command queue
  - cl_kernel kernel - kernel object
  - cl_uint work_dim - dimension of work-groups
  - const size_t *global_work_offset - global ID shift
  - const size_t *global_work_size - array of global work size
  - const size_t *local_work_size - array of local work size
  - cl_uint num_events_in_wait_list - events to wait for
  - const cl_event *event_wait_list - events to wait for
  - cl_event *event - fired event
- Possible return values
  - CL_SUCCESS or some error code

Kernel parameters

- dimension of work groups is usually: 1,2,3
- global work size: number of global work items (like gridsize*blocksize)
- local work size: number of work items in a work group (like blocksize)

Synchonize command queue

Flush queue

- Function name: clFlush
- Parameters
  - cl_command_queue command_queue - valid command queue
- Possible return values
  - CL_SUCCESS, CL_INVALID_COMMAND_QUEUE, CL_OUT_OF_RESOURCES, CL_OUT_OF_HOST_MEMORY
- Issues all previously queued OpenCL commands in command_queue to the device associated with command_queue
- There is no guarantee that they will be complete after clFlush returns

Finish queue

- Function name: clFinish
- Parameters
  - cl_command_queue command_queue - valid command queue
- Possible return values
  - CL_SUCCESS, CL_INVALID_COMMAND_QUEUE, CL_OUT_OF_RESOURCES, CL_OUT_OF_HOST_MEMORY
- Blocks until all previously queued OpenCL commands in command_queue are issued to the associated device and have completed
- It is a synchronization point

OpenCL

Basic terminology

Platform and device

Launch kernel

Memory management

Memory objects

Buffer

- Block of contiguous memory
- Hold general purpose objects
- Types of the values can be any of the built-in types
- Allocated by host API functions

Image

- Holds one/two or three dimensional images

Pipe

- Ordered sequence of data imes
- It has two endpoints
  - write endpoint for insert data
  - read endpoint for removing data
- At any time on kernel instance may write and another one may read from a pipe
- It is easy to implement the producer/consumer pattern

Create buffer object

OpenCL buffer object

- A buffer object stores a one-dimensional collection of elements
- Elements of a buffer object can be a scalar data type (such as an int, float), vector data type, or a user-defined structure

Create a new buffer object

- Function name: clCreateBuffer
- Parameters
  - cl_context context - valid OpenCL context
  - cl_mem_flags flags - bit field used to specify additional params
  - size_t size - size of buffer in bytes
  - void *host_ptr - host pointer (see flags)
  - cl_int *errcode_ret - error code
- errcode_ret values
  - CL_SUCCESS, CL_INVALID_CONTEXT, CL_INVALID_VALUE, CL_INVALID_BUFFER_SIZE, CL_INVALID_HOST_PTR, CL_MEM_OBJECT_ALLOCATION_FAILURE, CL_OUT_OF_RESOURCES, CL_OUT_OF_HOST_MEMORY
- Return value is a created buffer object (type of cl_mem)

Buffer flags

cl_mem_flags

- CL_MEM_READ_WRITE
  - memory object will be read and written by the kernel (default option)
- CL_MEM_WRITE_ONLY
  - memory object will be written only by the kernel
- CL_MEM_READ_ONLY
  - memory object will be read only by the kernel
- CL_MEM_USE_HOST_PTR
  - use memory referenced by host_ptr as the storage bits
  - host_ptr paramter must be not null
- CL_MEM_ALLOC_HOST_PTR
  - allocate memory from host accessible memory
- CL_MEM_COPY_HOST_PTR
  - allocate memory for the memory object and copy from host_ptr
- CL_MEM_HOST_WRITE_ONLY
  - the host will only write to the memory object
- CL_MEM_HOST_READ_ONLY
  - the host will only read from the memory object
- CL_MEM_HOST_NO_ACCESS
  - the host will not read or write the memory object

Manage buffer object

Retain and release buffer object

- Retain
  - Function: cl_int clRetainMemObject (cl_mem memobj)
  - Increments the buffer object reference count
- Release
  - Function: cl_int clReleaseMemObject (cl_mem memobj)
  - Decrements the buffer object reference count
  - After the memobj reference count becomes zero and commands queued for execution on a command-queue(s) that use memobj have finished, the memory object is deleted

Memory object query

- Function name: clGetMemObjectInfo
- Parameters
  - cl_mem memobj,
  - cl_mem_info param_name,
  - size_t param_value_size,
  - void *param_value
  - size_t *param_value_size_ret

Copying buffer objects (host to device)

Copy from host to device

- Function name: clEnqueueWriteBuffer
- Parameters
  - cl_command_queue command_queue - used command queue
  - cl_mem buffer - destination buffer
  - cl_bool blocking_write - blocking or non-blocking
  - size_t offset - shift
  - size_t size - number of bytes of data
  - const void *ptr - source host memory pointer
  - cl_uint num_events_in_wait_list - events must be completed
  - const cl_event *event_wait_list - events must be completed
  - cl_event *event - event to identify this copy
- Possible return values
  - CL_SUCCESS or some error code

Synchronization

- We can choose between blocking or non-blocking behavior
- We can use the given events to synchronize operations
  - wait for some events
  - fire a new event when operations is completed

Copying buffer objects (device to host)

Copy from host to device

- Function name: clEnqueueReadBuffer
- Parameters
  - cl_command_queue command_queue - used command queue
  - cl_mem buffer - source buffer
  - cl_bool blocking_read - blocking or non-blocking
  - size_t offset - shift
  - size_t size - number of bytes of data
  - const void *ptr - destination host pointer
  - cl_uint num_events_in_wait_list - events must be completed
  - const cl_event *event_wait_list - events must be completed
  - cl_event *event - event to identify this copy
- Possible return values
  - CL_SUCCESS or some error code

Synchronization

- We can choose between blocking or non-blocking behavior
- We can use the given events to synchronize operations
  - wait for some events
  - fire a new event when operations is completed

Copying between buffers

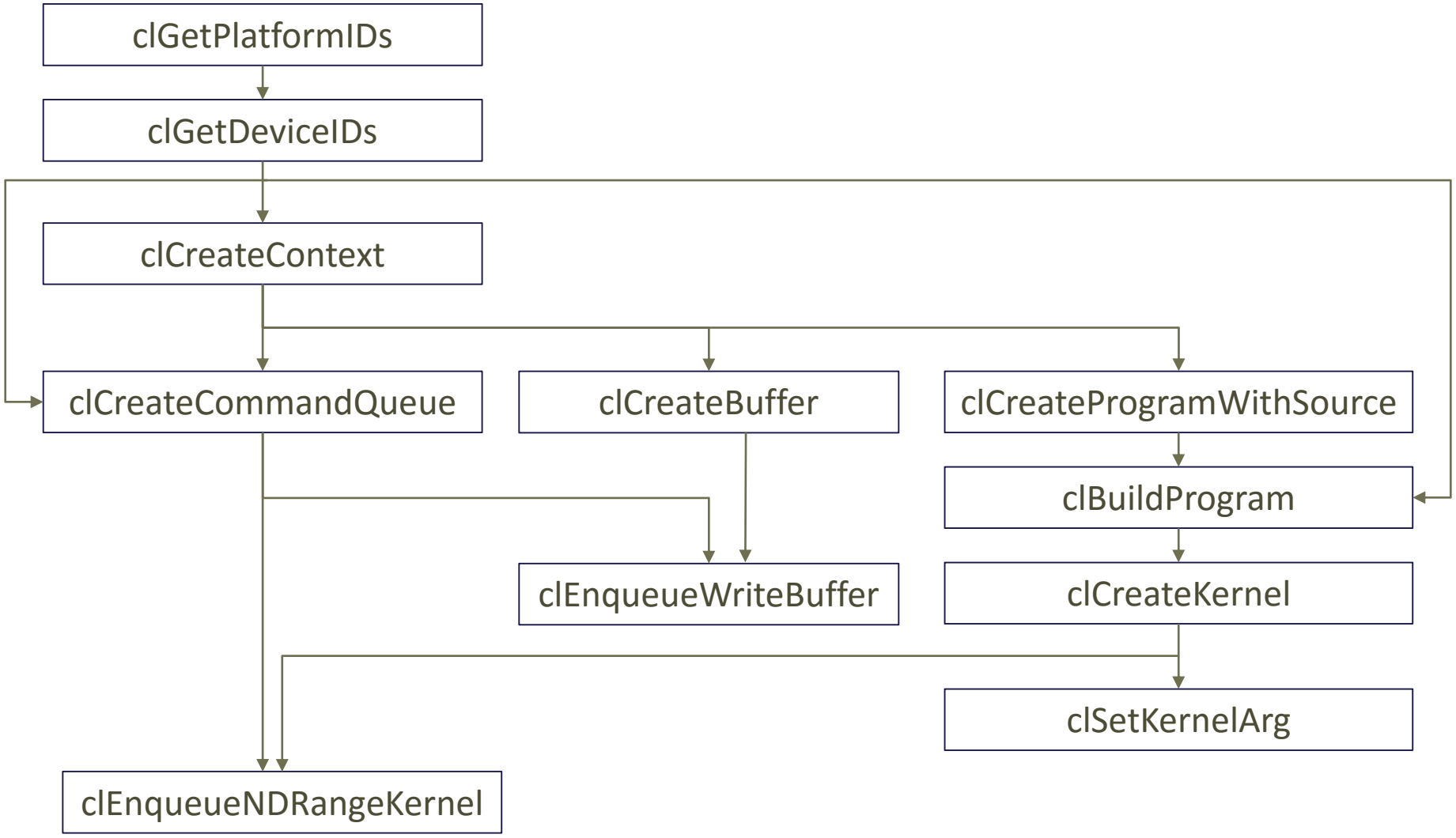
Copy from buffer to buffer

- Function name: clEnqueueCopyBuffer
- Parameters
  - cl_command_queue command_queue - used command queue
  - cl_mem src_buffer - source buffer
  - cl_mem dst_buffer - destination buffer
  - size_t src_offset - source shift
  - size_t dst_offset - destination shift
  - size_t size - number of bytes of data
  - cl_uint num_events_in_wait_list - events must be completed
  - const cl_event *event_wait_list - events must be completed
  - cl_event *event - event to identify this copy
- Possible return values
  - CL_SUCCESS or some error code

Synchronization

- We can use the given events to synchronize operations
  - wait for some events
  - fire a new event when operations is completed

Full process to run a kernel



Local memory

OpenCL local memory

- A memory region associated with a work-group and accessible only by work-items in that work-group
- Local memory is included within the generic address space that includes the private and global address spaces

Static allocation

- Using the keyword: __local

Dynamic allocation

- Kernel code

```
__kernel void kernel(__local float *shr_mem)
```

- Host code

```
clSetKernelArg(kernel, 1, 32*sizeof(float), NULL);
```

- In this case every block will have an allocated array in the local memory referenced by the shr_mem pointer