BUFFERS AND THE HOST-SIDE MEMORY MODEL

OPENCL 2.0 UNIVERSITY TOOLKIT

Zhongliang Chen and Yash Ukidave,
Northeastern University Computer Architecture Research Lab
with
Perhaad Mistry and Dana Schaa, AMD

INSTRUCTOR NOTES

AMD

- This lecture details memory object creation, reading, and writing
  - Buffers are used for examples
- The OpenCL specification has very few specific requirements for data allocation and movement. While programmers can conceptualize that writing and reading buffers moves data to and from the device, this is not explicitly required. The memory management examples attempt to show some of the subtleties of real implementations.

TOPICS

- Memory object creation (buffers as an example)
  - Memory flag options
- Writing and reading buffers
- Memory object management
- Memory object migration
- Host-accessible memory

BUFFERS

AMD

- Memory objects are used for passing large data structures to OpenCL kernels
- The most straight-forward object is a buffer
  - A buffer is a contiguous sequence of addressable of elements similar to a C array
- Based on the memory flags provided, a host pointer can be optionally supplied to initialize the buffer, or even provide storage for the buffer
- A buffer object is created using the following function:

```
cl_mem  buffer = clCreateBuffer (
        cl_context context,  // Context object
        cl_mem_flags flags,  // Memory flags
        size_t size,         // Bytes to allocate
        void *host_ptr,      // Host data
        cl_int *errcode)     // Error code
```

MEMORY FLAGS

AMD

- Memory flag field in clCreateBuffer() allows us to define attributes of the buffer object

Memory Flag	Behavior
CL_MEM_READ_WRITE	Specifies the types of accesses that are allowed by a kernel
CL_MEM_WRITE_ONLY	
CL_MEM_READ_ONLY	
CL_MEM_USE_HOST_PTR	Use the host pointer as storage for the buffer. Implementations can still cache the contents to device memory and use the cached copy when a kernel executes.
CL_MEM_ALLOC_HOST_PTR	Allocates buffer storage in host accessible memory.
CL_MEM_COPY_HOST_PTR	Initialize the buffer with data from referenced by host_ptr.
CL_MEM_HOST_WRITE_ONLY	Specifies the access types allowed by the host towards the memory object.
CL_MEM_HOST_READ_ONLY	
CL_MEM_HOST_NO_ACCESS	

BUFFERS

AMD

- The following code creates a read-only buffer and initializes it with data from a host array
  - Assume that `context` is a valid OpenCL context

```
cl_int err;
int a[16];

cl_mem newBuffer = clCreateBuffer(
    context,
    CL_MEM_READ_ONLY | CL_MEM_USE_HOST_PTR,
    16*sizeof(int),
    a,
    &err);

if( err != CL_SUCCESS ) {
    // Handle error as necessary
}
```

WRITING BUFFERS

AMD

- OpenCL provides commands to read or write data from a buffer
  - The use of a command queue also allows the runtime to copy the buffer to that device if it desires to do so
  - Generates an event for dependencies, or can be specified as a blocking call
- Once the command completes the host pointer can be reused
  - A programmer can conceptualize that the data storage of the buffer object resides on the device after the call completes, though this is not explicitly required by the OpenCL specification

```
cl_int clEnqueueWriteBuffer (
    cl_command_queue queue,              // command queue
    cl_mem buffer,                       // buffer object
    cl_bool blocking_write,              // blocking flag
    size_t offset,                       // offset into buffer to write
    size_t cb,                           // size of data to write
    void *ptr,                           // pointer to source data
    cl_uint num_in_wait_list,            // number of events to wait for
    const cl_event *  event_wait_list, // array of events to wait for
    cl_event *event)                     // event for this command
```

WRITING BUFFERS

AMD

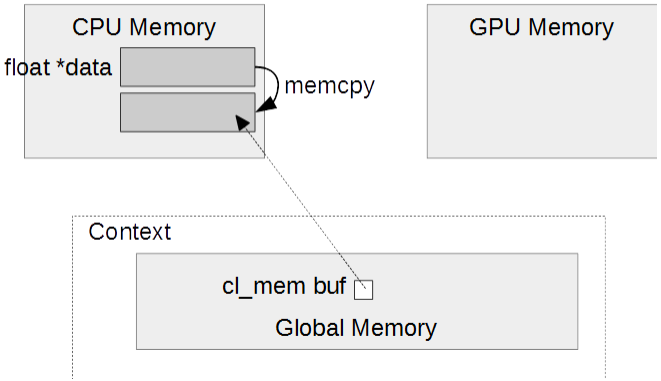
- An example showing buffer creation and initialization

```
cl_int err;
int a[16];

for (int i = 0; i < SIZE; i++) {
    a[i] = i;
}

// Create the buffer
cl_mem buffer = clCreateBuffer(context, CL_MEM_READ_WRITE, SIZE*sizeof(int),a, &err);

if( err != CL_SUCCESS ) {// Handle error as necessary }

// Initialize the buffer
err = clEnqueueWriteBuffer (
    queue,
    buffer, // destination
    CL_TRUE, // blocking write
    0, // don't offset into buffer
    SIZE*sizeof(int), // number of bytes to write
    a, // host data
    0, NULL, // don't wait on any events
    NULL); // don't generate an event
```

WRITING BUFFERS

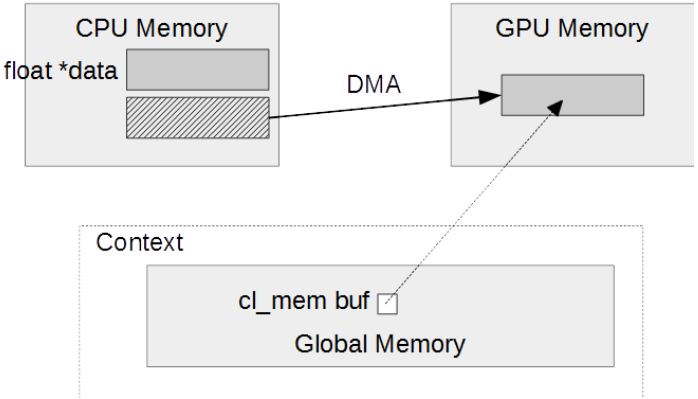
- The following example shows that we can create and initialize a buffer, and use it in a kernel, without explicitly writing the buffer

```
cl_mem buf = clCreateBuffer(…,
   CL_MEM_COPY_HOST_PTR,data,…);
```



a) Creating and initializing a buffer in host memory (initialization is done using CL_MEM_COPY_HOST_PTR).

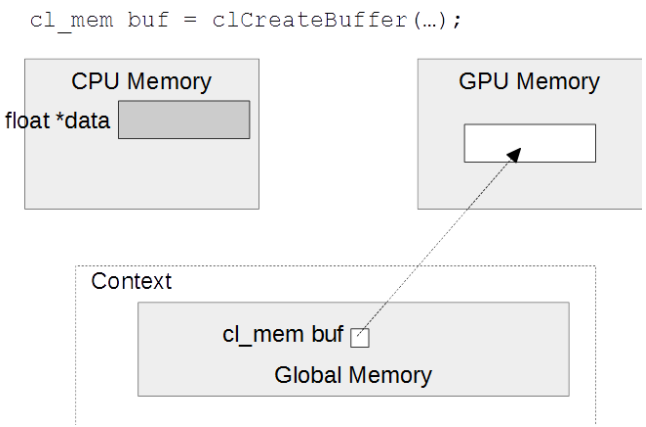
```
clSetKernelArg(…, &buf);
clEnqueueNDRangeKernel(gpuQueue,…);
```



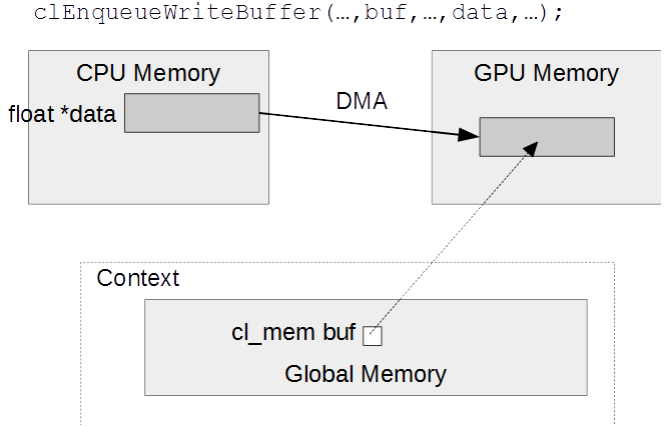
b) Implicit data transfer from the host to device prior to kernel execution. The runtime could also choose to have the device access the buffer directly from host memory.

WRITING BUFFERS

- As an alternative, a runtime may decide to create the buffer directly in device memory



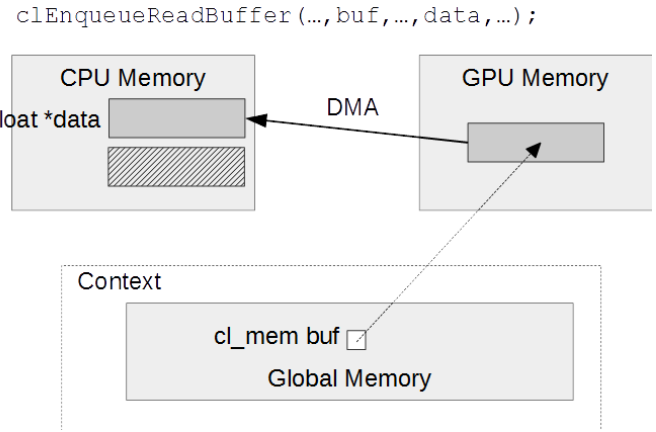
a) Creating a buffer in device memory (at the discretion of the runtime)



b) Copying host data to the buffer directly in GPU memory

READING BUFFERS

- The complimentary call to writing a buffer, reads the buffer back to host memory
  - The following diagram assumes that the buffer data had been moved to the device by the runtime



c) Reading the output data from the buffer back to host memory (continued from the previous slide)

MIGRATING MEMORY OBJECTS

AMD

- OpenCL provides an API call to migrate memory objects between devices
  - Unlike reading and writing a buffer this call guarantees that memory objects will be located on the device when the command completes
  - Allows multiple memory objects to be migrated with a single command

```
cl_int clEnqueueMigrateMemObjects (cl_command_queue command_queue,
                                   cl_uint num_mem_objects,
                                   const cl_mem *mem_objects,
                                   cl_mem_migration_flags flags,
                                   cl_uint num_events_in_wait_list,
                                   const cl_event *event_wait_list,
                                   cl_event *event)
```

HOST-ACCESSIBLE MEMORY

AMD

- When creating a memory object, flags can specify that it should be created in host-accessible memory
  - I.e., requires allocation in a place that can be mapped into the host's address space
- CL_MEM_ALLOC_HOST_PTR
  - Simply tells the runtime to create the buffer in host-accessible memory
- CL_MEM_USE_HOST_PTR
  - Tells the runtime to use the supplied host pointer as storage for the buffer
    - Can be used to prevent redundant copies of data, especially in APU environments
- For both options, it is reasonable to assume that implementations will have the device access the buffer from host (CPU) memory
  - This is commonly referred to as *zero-copy* memory
  - This is not explicitly required by the OpenCL specification

HOST-ACCESSIBLE MEMORY

AMD

- AMD-specific treatment of the flags
- CL_MEM_ALLOC_HOST_PTR and CL_MEM_USE_HOST_PTR
  - If devices support virtual memory, storage will be created as pinned (non-pageable) host memory, and accessed as zero-copy data
  - Without virtual memory, storage will be allocated on the device
- CL_MEM_USE_PERSISTENT_MEM_AMD
  - Vendor-specific extension
  - Accesses to this memory object from the host will occur directly from device memory
- When specifying where data should be stored, make sure you understand the performance implications of your choices

MAPPING DATA TO HOST MEMORY

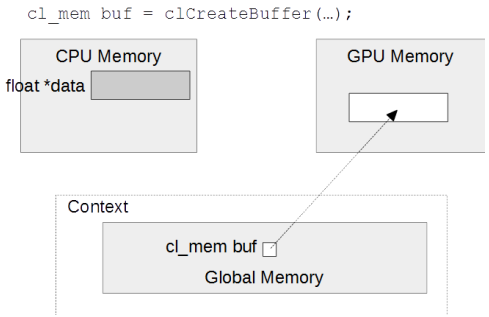
- OpenCL provides API calls to map and unmap memory objects from the host's memory space
- When a memory object is mapped, a pointer valid on the host will be returned
  - The parameters to the map function should look similar to reading/writing a buffer

```
void * clEnqueueMapBuffer (cl_command_queue command_queue,
                           cl_mem buffer,
                           cl_bool blocking_map,
                           cl_map_flags map_flags,
                           size_t offset,
                           size_t size,
                           cl_uint num_events_in_wait_list,
                           const cl_event *event_wait_list,
                           cl_event *event,
                           cl_int *errcode_ret)
```

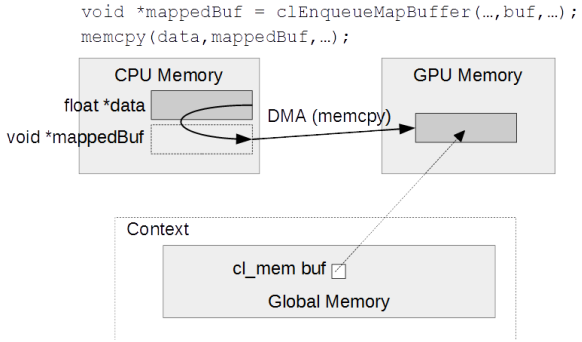
```
cl_int clEnqueueUnmapMemObject (cl_command_queue command_queue,
                                cl_mem memobj,
                                void *mapped_ptr,
                                cl_uint num_events_in_wait_list,
                                const cl_event *event_wait_list,
                                cl_event *event)
```

MAPPING DATA TO HOST MEMORY

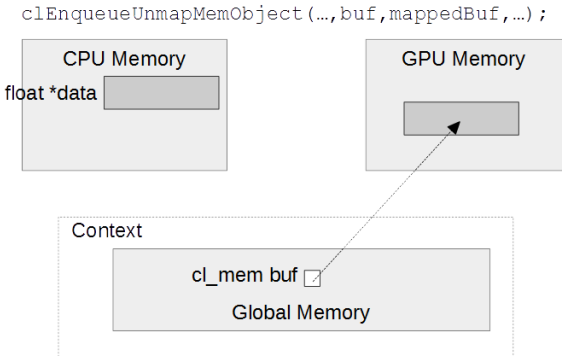
- The example below shows one possible scenario for the map and unmap functions



a) Create an uninitialized buffer in device memory



b) Map the buffer into the host's address space. Access the buffer via a host pointer.



c) Unmap the buffer from the host's address space

SUMMARY

AMD

- Memory objects are created and managed by the host using OpenCL API calls
- Options provided during creation can be used to describe the programmer's intent to the runtime
- Remember that it is ultimately up to the runtime to determine where data is allocated and resides, and when and where it should be transferred