Recitation 7

Announcements

- This assignment is optional.

Assignment 6

- Some theory.
- Mandelbrot using OpenCL.
- OpenCL uses the same programming model as CUDA, this assignment is therefore very similar to PS5.

OpenCL vs CUDA

- Both OpenCL and CUDA are used to do the same thing, in a similar way.
- CUDA is only for NVIDIA GPUs, OpenCL is an open standard, can be run on multiple platforms (GPUs, CPUs FPGAs).

Setup

OpenCL

```
clGetPlatformIDs(...);
clGetDeviceIDs(...);
context = clCreateContext(...);
queue = clCreateCommandQueue(...);
source = load_program_source(...);
program = clCreateProgramWithSource(...);
err = clBuildProgram(...);
kernel = clCreateKernel(...);
```

- Because it's cross platform, and can do multiple devices of different kinds, OpenCL requires more set up code.
- I'll include most of this in the code handed out.

OpenCL

- The kernel are compiled at run time (so you'll get compilation errors when you run the program.
- You'll then get a `cl_kernel` variable which can be used later to start the kernel.
- OpenCL is more explicitly asynchronous than CUDA. Most functions simply enqueue something in the command queue and return imedeately.

Memory example

```
float *A, *B;
...
cl_mem A_d = clCreateBuffer(context,
  CL_MEM_READ_WRITE, size, NULL, &err);
err = clEnqueueWriteBuffer(queue, A_d, CL_TRUE,
  0, size, A, 0, NULL, NULL);
...
err = clEnqueueReadBuffer(queue, A_d, CL_TRUE,
  0, size, B, 0, NULL, NULL);
```

Allocate memory, transfer from host to device, transfer from device to host.

Memory allocation

OpenCL

```
cl_mem clCreateBuffer (cl_context context,
  cl_mem_flags flags,
  size_t size,
  void *host_ptr,
  cl_int *errcode_ret)
```

- Allocates memory on the device/GPU, equivalent to cudaMalloc().
- Most relevant flags are `CL_MEM_READ_WRITE` and `CL_MEM_COPY_HOST_PTR`

Memory transfer

OpenCL

```
cl_int clEnqueueReadBuffer (
   cl_command_queue command_queue,
  cl_mem buffer,          // Device buffer
  cl_bool blocking_read,
  size_t offset,
  size_t cb,
  void *ptr,              // Host buffer
  cl_uint num_events_in_wait_list,
  const cl_event *event_wait_list,
  cl_event *event)
```

- Used to transfer data from device to host memory.
- For this assignment you should use blocking OpenCL transfers.

Memory transfer

OpenCL

```
cl_int clEnqueueWriteBuffer (
    cl_command_queue command_queue,
   cl_mem buffer,              // Device buffer
   cl_bool blocking_write,
   size_t offset,
   size_t cb,
   const void *ptr,            // Host buffer
   cl_uint num_events_in_wait_list,
   const cl_event *event_wait_list,
   cl_event *event)
```

- Transfer data from host to device memory.
- The event arguments for both read and write can be set to just 0/NULL

Launching kernels, OpenCL

OpenCL

```
err = clSetKernelArg(kernel, 0,
  sizeof(cl_mem), &A_d);
err = clSetKernelArg(kernel, 1,
  sizeof(cl_mem), &B_d);
const size_t g_ws = {256,256};
conts size_t l_ws = {16,16};
err = clEnqueueNDRangeKernel(queue, kernel,
  2, NULL, g_ws, l_ws, 0, NULL, NULL);
```

- Kernel arguments must be set with separate functions.
- `g_ws` is the total number of *work-items*, `l_ws` is the number of *work-items* per *work-group*.

Setting arguments

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

Used to set the argument of kernels.

```
cl_int clEnqueueNDRangeKernel (
    cl_command_queue command_queue,
   cl_kernel kernel,
   cl_uint work_dim,
   const size_t *global_work_offset,
   const size_t *global_work_size,
   const size_t *local_work_size,
   cl_uint num_events_in_wait_list,
   const cl_event *event_wait_list,
   cl_event *event)
```

Used to start a kernel.

Finding thread ID

OpenCL

```
int id = get_global_id(0);
int l_id = get_local_id(0);
```

- Functions/built in variables to find global/local id as well as local and global sizes.
- These examples are for 1D grids.

Function qualifiers

- Kernels should use the `__kernel` qualifier (like `__global__` in CUDA.
- No qualifier needed for device functions. (E.g. no `__device__`).

Memory spaces

- OpenCL implements the memoryspaces global, local, constant and private.
- Variables in these should be decalred with `__global, __local, __constant, __private` respectively.
- These memoryspaces are logical only. OpenCL only specifies how they can be accessed etc. not how they should be physically implemented (since OpenCL should be able to run on many platforms).
- Local is the same shared on CUDA (both physically and logically).
- Private is the default for variables in kernels, and the `__private` is not needed.

Error handling

- Most/all OpenCL functions return an error code, (either directly or through an argument).
- I'll provide a function which translates this into a error string.

Compilation

OpenCL

```
$ gcc -std=c99 -o test test.c -lOpenCL
$ ./a.out
```

- The host code is put in a .c file, the device code in a .cl file.
- The OpenCL host code is compiled with a regular C compiler. The device code is compiled by the OpenCL implementation at runtime.