OpenCL Exercise

In this exercise, we will look at OpenCL, which is relavant for C programmers. There is no Fortran equivalent.

Introduction

Navigate to find the template source code in `exercises/opencl_intro/intro.c`. There should be an accompanying kernel template `kernel.c`.

The template source file is marked with the sections to be edited, e.g.

/* Part 1A: allocate device memory */

Please read on for instructions. Where necessary, it may be helpful to refer to the OpenCL reference guide https://www.khronos.org/files/opencl22-reference-guide.pdf

1) Copying Between Host and Device

The exercise is a simple example in which an array of integers is multiplied by a constant (-1). This should allow you to contrast two approaches: OpenCL and CUDA.

In the OpenCL template `intro.c`, some of the platform and device management has been provided so that you should be able to concentrate on device memory allocation, transfer between host and device, and the kernel itself.

Starting from the `intro.c` template:

- 1A) Allocate memory for the array on the device using clCreateBuffer(). The appropriate device reference is d_a (check its declaration). You will need to identify the OpenCL context and the size of allocation (sz). Check the OpenCL reference card for the arguments to clBufferCreate() if unsure.
- 1B) Copy the array h_a on the host to d_a on the device. The OpenCL command queue is varaible queue.
- 1C) Copy d_a on the device back to h_out on the host.
- 1D) Release the device memory d_a.

Compile and run the code

Note here we are using nvcc to compile and OpenCL program. We must also specify that the executable is linked against the OpenCL library via -lOpenCL on this system.

```
# Execute this command to compile the code.
$ make

# Execute this command to submit the job to the queue system.
$ qsub submit.sh
```

The output (the contents of the h_out array) or any error messages will be printed as output. So far the code simply copies from h_a on the host to d_a on the device, and then copies d_a back to h_out, so the output should be the initial content of h_a, that is, the numbers 0 to 255.

2) Executing the Kernel

Now we will extend the intro.c file to run the kernel.

- 2A) Configure the global number of work items and the local number of work items for this problem. Note that the number of items per work group has been assigned as a constant to THREADS_PER_WORK_GROUP. You will need to declare the kernel argument, and execute it with the appropriate OpenCL calls.
- 2B) Look at kernel template kernel.c. Implement the computation required (multiply each element of the array by -1) using the OpenCL function get_global_id()

Compile and run the code by executing the commands above as before. This time the output should contain the result of negating each element of the input array. Because the array is initialised to the numbers 0 to 255, you should see the numbers 0 to -255 printed.

Additional tasks

- (If you haven't found out yet.) What happens if you introduce a programming (compilation) error into the kernel code? Where is the error message coming from?
- Look through the rest of the template intro.c to see what is required to initise OpenCL and read in the kernel.

Keeping a copy of your work

The templates for the exercise and a solution are provided in the course repository.

If you wish to keep a copy of your particular soultion, please use the scp command to obtain a copy of the relevant files.