OS (H) Assessed Exercise: OpenCL Host Programming – Part II

Wim Vanderbauwhede

1 Aim

The aim of this coursework is to create a simple host-side driver routine `run_driver()` to interact with an OpenCL-compliant device (e.g. a GPU). This `run_driver()` routine is called from a multithreaded testbench which will take care of initialisation and shutdown of the device, creation of the data to be sent to the device and validation of the returned result. Source code for such a testbench will be provided for the second part of the assignment.

2 What to submit

NOTE: In this section, the use of the words *must* or *should* means you will lose marks if you don't do this.

For the second part of the assignment, you have to write the code for the `run_driver()` based on the provided skeleton and the required OpenCL and POSIX pthread mutex API calls to protect the critical section. You are encouraged to read the relevant OpenCL API specification in detail.

- You *must* start from the provided code and you should *only add your own code* in `driver.c`, you *must* not modify or remove any part of the provided code.

- You *must* use the variable `err` for the error code for *all* API calls and report the error to *stderr* using `fprintf`.

- If you use `printf` statements in your code, they *must* be enclosed by an `#ifdef VERBOSE` ... `#endif` guard.

- You *must* submit this code in a *gzipped tar archive* (other formats will *not* be accepted) through the Moodle submission system, and the filename *must* be *<your matric + 1st char of your name in lowercase>.tgz*, so for example if your matric number is *1107023m* then your archive *must* be named *1107023m.tgz*. This archive *must* contain a single folder named *<your matric + 1st char of your name in lowercase>*. This folder *must* contain following files:

- `driver.h` as provided
- `driver.c` in which you must implement the functionality for the `run_driver` subroutine
- `testbench.c` as provided
- `firmware.cl` as provided
- `Makefile` as provided

3   What is provided

On Moodle you will find:

- The official specification of the OpenCL API version 1.2
- An archive `opencl_driver_skeleton.tgz` containing a the files listed above with a skeleton for `run_driver()`.

4   How to test your OpenCL code

There are three possible ways to test your OpenCL code:

1. Install OpenCL on your Raspberry Pi. For instructions on how to install OpenCL on your Raspberry Pi, see this tutorial.
2. Install OpenCL on your own machine. To install OpenCL locally, do the following:

   - Download the AMD OpenCL SDK 64 bit from http://developer.amd.com/tools-and-sdks/opencl-zone/amd-accelerated-parallel-processing-app-sdk/
   - Unpack the archive
     $ tar -jxf AMD-APP-SDKInstaller-v3.0.130.135-GA-linux64.tar.bz2
   - Run the installer and choose the defaults
     $ ./AMD-APP-SDK-v3.0.130.135-GA-linux64.sh
   - Log out and log back in or open a new shell (the installer modifies your .bashrc)
   - Now you can use the provided Makefile, testbench and driver code skeletons and kernel source to build and test your OpenCL application.

3. Use the OpenCL pre-installed on the lab machines. The OpenCL SDK is available in `/opt/intel/opencl`.

5 Marking

Your code will be marked using a test script. This script will test if your code builds correctly and runs correctly for a number of use cases. I will not modify the testbench code but may run modified firmware and change the number of threads and the buffer size. The script will also perform source code analysis to see if you have used the correct API calls, control structures etc.

The coursework is marked out of 60, the marking scheme is as follows:

- Compilation, running and passing tests 20/60

- Code correctness 40/60, based on

    - Correct use of pthread API calls
    - Correct use of OpenCL API calls
    - Correct error handling