OpenCL Hands-on

SPD Course

2017-18

Massimo Coppola

OpenCL on Titanic

- You should all have received credentials to log in on titanic
  - Never forget to check for reservations
  - Never forget to check GPU temperature when running the GPU for more than a minute
  - **tail /var/log/gputemp** is your friend
- As part of the CUDA 8.0 installation
  - Only supports OpenCL 1.2 officially
  - No examples already installed
- Example sources (see course page for URL)
  - Nvidia SDK
  - HandsOnOpenCL (open source on Github)

HandsOnOpenCL exercises

- Git clone from repository or
- Find the archive in the /home directory
  - **Exercises-Solutions-master.zip**
- You need to export variables pointing at the OpenCL path on titanic

```
export CPATH=/usr/local/cuda/include/
export LD_LIBRARY_PATH=/usr/local/cuda/lib64/
```

  Or set those variables in the main makefiles
- Get into the exercises directories and try
  - Check the Readme
  - Make & run, in the simple cases
  - Edit the source to complete the exercise, test it

HandsOnOpenCL exercises

- Makefiles are such as they will usually select the default CL device (e.g. the CPU). So again:

```
export DEVICE=CL_DEVICE_TYPE_GPU
```

- Use either the C or C++ version
  - your choice when both sources are available
- 01 test it!
- 02, 03 test the code, examine it
- 04, 05, 06, 07, 08, you can do them easily
- 10 try it: CPU and GPU as parallel CL devices
- 11 try it at home, it requires time
- 12 is easy

Some notes on OpenCL C++ bindings

- Context creation can set up a default device for following CL calls
  - `cl::Context context(DEVICE);`
- Buffer constructors accept iterators as inputs
- `cl::make_kernel` template function
  - Takes a program object and the kernel name as arguments
  - Template parameters are the types of corresponding kernel parameters
  - Returns a functor (it overloads the () operator)
    - The functor requires a cl::EnqueueArgs object and the necessary number of parameter objects (e.g. cl::Buffers)
    - It can be used right away, or stored in an *auto* variable and called later on to enqueue that kernel for execution

C++: error logs of kernel compilation

- When developing new kernels, prefer loading them from external files
- Here exceptions are enabled via **`#define __CL_ENABLE_EXCEPTIONS`**
- Disable automatic build to access the build log
- Edit the CL kernel and relaunch the program
- A bit easier in C or with exceptions disabled: check the error code at build

```
cl::Device device = …
cl_int builderror;
cl::Program program(context, util::loadProgram("mYKERN.cl"), false, &builderror);
std::cerr << "cl::program error " << builderror << std::endl;

try {  program.build(); }
catch (cl::Error& err) {
  if (err.err() == CL_BUILD_PROGRAM_FAILURE) {
    cl_build_status status = program.getBuildInfo<CL_PROGRAM_BUILD_STATUS>(device);
    std::cerr << "BUILDSTATUS " << status << std::endl;
    std::string buildlog = program.getBuildInfo<CL_PROGRAM_BUILD_LOG>(device);
    std::cerr << "Build log" << std::endl << buildlog << std::endl;
  }   else {throw err;}
}
```

Working / Testing the Exercise problems

- When stuck
  - Try harder, check in small steps what is happening in your code
  - If the issue is in the CL kernel, check build logs and buffer initializations/copies
  - Compare your code with the provided solution
- When testing performance
  - Enable multiple executions of the same test
    - what is the timer resolution?

Exercise 2 - OpenCL K-means

- You are provided a reference code executing k-means from text data on file
  - Code archive in /home ; the kernel shall performs one pass on the data array
- Modify the skeleton by converting the sequential code into a CL kernel
  - can we make the code more compact?
    - Vector operations on data, centroid groups
  - What evolutions can be performed on the code?
    - Add a real convergence criterion (not just fixed iteration num)
    - Code optimizations on the kernel
    - Allow the kernel to make more than on algorithm pass
    - Asynchronous computations on the GPU
      - multiple sets of centroids, independent searches
    - Asynchronous computations on GPU and CPU
    - Different general optimization search strategies
      - Choice of new initial centroid sets
      - Early termination if centroid sets seem to converge to the same or already known optimum (requires checking for permutations!)
    - Can these options be implemented exploiting TBB?