Lab session 1

- **Login:** `ssh username@oakley.osc.edu`.
- **Ensure CUDA is available:** `module load cuda`.
- **Acquire and unpack example source code:**

```
mkdir CUDAex
cd CUDAex
wget http://developer.nvidia.com/sites/default/files/akamai/cuda\
/files/cuda_by_example.zip
unzip cuda_by_example.zip
```

- **Compile find-the-GPUs example:**

```
cd chapter03
nvcc -o enumGPUs enum_gpu.cu
```

Batch submission

- **First run the enum program in batch; this requires a submission script:**

```
#PBS -l walltime=0:10:00
#PBS -l nodes=1:ppn=12:gpus=2
#PBS -N my_job
#PBS -S /bin/bash
#PBS -j oe
module load cuda
pbsdcp $HOME/CUDAex/chapter03/enumGPUs $TMPDIR
cd $TMPDIR
./enumGPUs
```

- **Submit and check status:**

```
qsub subenumscript
qstat | grep username
```

- **Other useful commands:**

```
showstart jobname
qpeek jobname
qdel jobnumber
```

Some comments

- **Execution node has access to your home directory, but this is slow and will make you unpopular if abused.**
- **Move input data and executables to $TMPDIR before using them with** `pbsdcp` **command.**
- **Log output will go in file** `JobName.oJobNumber` **in the directory from which you run qsub.**
- **If your program writes to a file, or otherwise has output that's not going to the logfile, copy it from $TMPDIR to $HOME at the end of your batch script.**
- **Batch queue can be kind of slow. This is fine if you're submitting something that will take three hours; go have lunch. If it'll take thirty seconds and then you change the code, it's annoying to wait ten minutes per submission.**

Interactive sessions

- **For quick testing, small jobs, anything that requires rapid feedback, use an *interactive batch* session instead.**
- **In effect we are logging in to an execution node:**

```
qsub -I -l nodes=1:ppn=6:gpus=1 -l walltime=01:00:00
```

- **Now it works just like any other login. Load modules, compile, run. But note that access to your home directory is slow! Copy things across just as before.**

```
module load cuda
cd $TMPDIR
cp $HOME/CUDAex/chapter03/enumGPUs .
./enumGPUs
```

- **When done with an interactive session, do `exit`.**

Coding exercise

- **Copy incomplete program from `~ucn1122/cudacourse/lab1/exercise1.cu`.**
- **Part 1 (just to get the fingers limber and the brain back in coding gear): In CPU code, multiply two vectors storing the result in a third; sum the results; print the sum; and take the time of the multiplication and sum.**
- **Part 2 (now we use the GPU): Use `cudaMalloc` and `cudaMemcpy` to get memory space on the GPU and to copy the two data vectors across.**
- **Part 3: Implement the device-side function `device_vector_mult`. Notice that this function only deals with one data item per call!**
- **Part 4: Launch your new kernel.**
- **Part 5: Use `cudaMemcpy` in the other direction to put the device-side results into the `results` vector.**
- **Part 6: Repeat the sum-over-array, timing, and printing-the-results bits from before.**
- **Part 7: Write device-side reduction code, and time this operation separately so you can compare it with the move-to-host, sum-on-host approach above.**
- **Part 8: Run your program with several different values of `sizeOfVector`. Notice that this is a command-line parameter, no need to recompile. Make a table of how the execution time changes for the CPU and GPU code.**

- **Note: To compile your code, do**

  ```
  module load cuda
  nvcc -o dotproduct exercise1.cc
  ```

  noting that you only need to load the module the first time. To run the code, make an interactive batch session as shown for the `enumGPUs` program above.