DUE DATE: **MON, OCT 6 2014 BY 5:00 PM (EST)**

Heterogeneous and Cloud Computing
Exercise #8

Max Points: 20

Note: If you are using your personal machine then it is best to install a Linux virtual machine on your computer and use the Linux virtual machine.

You should save and rename this document using the naming convention `MUid_Exercise8.docx` (example: `raodm_Exercise8.docx`).

Objective: The objective of this exercise is to gain some familiarity with:

- Loading and running OpenCL kernel from a file.
- Run OpenCL kernel on machine with a Xeon Phi accelerator.

Submission: Once you have completed this exercise, upload:

1. The C++ program completed as part of this exercise named with the convention *`MUid`*`_Ex8.cpp`.
2. The OpenCL kernel developed as part of this exercise in a file named `MUid_toggle.cl`.

Fill in answers to all of the questions as directed. For some of the questions that require outputs to be indicated, you can simply copy-paste appropriate text from the shell/PuTTY window into this document. You are expected refer to `LinuxEnvironment.pdf` document available in Handouts folder off Niihka. You may discuss the questions with your instructor.

Name:

Task #1: Convert program to use OpenCL Kernel

Copy the supplied `raodm_ex4.cpp` to a file named `MUid_ex8.cpp` to serve as starter code. In this exercise you are expected to convert the toggle method in the supplied C++ program (solution from Exercise #4) to run as an Open CL kernel, with the following requirement:

Requirement: The OpenCL kernel should be placed and loaded from a separate file named `toggle.cl`. In order to compile the program into a kernel, you will need to load it from the file into a string and then use it to build the program and create a kernel.

Tips: It would be best to streamline your solution by organizing your program into several cohesive helper methods. The reference solution uses the following helper methods:

```
cl::Program
getProgram(std::vector<cl::Device>& deviceList, cl::Context& context,
           const std::string& fileName) throw (cl::Error);
```

```
cl::Kernel
getKernel(cl::Context& context, cl::CommandQueue& queue,
          const std::string& fileName = "toggle.cl",
          const std::string& entryFunc = "toggle") throw (cl::Error);
```

Sample outputs

The expected outputs are shown below (note that the output from the standard C++ version and the OpenCL version of the program should be identical):

```
$ ./raodm_ex8 small_test.txt t
this is a small TexT file
used for TesTing. For Timings
generaTe a large TexT file using
soluTion for exercIse #3.
```

```
$ ./raodm_ex8 small_test.txt t i m q
thIs Is a sMall TexT fIle
used for TesTIng. For TIMIngs
generaTe a large TexT fIle usIng
soluTIon for exercise #3.
```

```
$ ./raodm_ex8 small_test.txt t i m q t i m > out.txt
$ diff small_test.txt out.txt
```

Part #2: Code Review

Once you have completed the previous part of the exercise, share your solution with your instructor and have your instructor verify your solution.

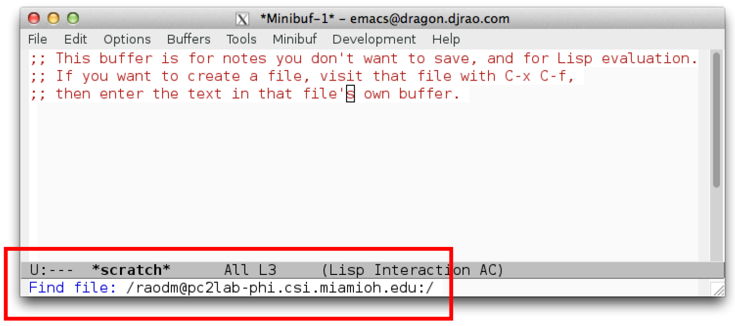
Part #3: Run on Xeon Phi

Once you have completed the previous part of the exercises try running your OpenCL program using the Xeon Phi accelerator available on PC2Lab's (http://pc2lab.cec.miamiOH.edu/) Xeon Phi research server as shown below:

```
$ ssh -X pc2lab-phi.csi.miamioh.edu
```

Tips:

1. The easiest way to copy, compile, and run the program on the server would be to do it from `emacs`, as `emacs` is fully POSIX compliant. In `emacs` open a file directly on the remote machine using CTRL+X CTRL+F and typing the UNC path in the form: `/user@host:` as shown in the adjacent screen shot. Now, when you compile, run, and debug the program it will run on the remote machine rather than on the local machine.


Minibuf-1 – emacs@dragon.djrao.com
File Edit Options Buffers Tools Minibuf Development Help

```
;; This buffer is for notes you don't want to save, and for Lisp evaluation.
;; If you want to create a file, visit that file with C-x C-f,
;; then enter the text in that file's own buffer.

U:---  *scratch*      All L3     (Lisp Interaction AC)
Find file: /raodm@pc2lab-phi.csi.miamioh.edu:/
```

2. To determine the OpenCL configuration of `pc2lab-phi` you can use the solution from Exercise #6 (`platform_info.cpp`) so that you can select the accelerator device on the appropriate platform.

3. You can use the solution for Exercise #3 off Niihka to generate a large data file for testing.

4. Note that `pc2lab-phi` has only one Xeon Phi accelerator in it. Consequently, you have to take turns to run your program on the device. In order to execute any OpenCL program, you will have to queue up your jobs on `pc2lab-phi` using the `qexec.sh` as shown in the example below:

```
$ qexec.sh /usr/bin/time -v ./raodm_ex8 100mb_file.dat a e i o u x q r z > /dev/null
```

Submit files to Niihka

Once you have completed this exercise, upload:

1. The C++ program completed as part of this exercise named with the convention `MUid_Ex8.cpp`.
2. The OpenCL kernel developed as part of this exercise in a file named `MUid_toggle.cl`.