CS 241 Honors
HW 2: Parallel Programming++

Assigned: October 14, 2015
Due: TBD

1 Getting the Assignment

Files for this homework have been pushed to your GitHub repositories. If you have not already done so, clone a local copy of your repo:

```
$ git clone https://github.com/{your github username}/{your netID}
```

If you are on EWS, you may need to do this over ssh, which requires generating an ssh key and adding it to your GitHub account. See `https://help.github.com/articles/generating-ssh-keys/#platform-linux` for instructions on how to do this.

If you have already cloned your repo to your machine, `cd` into your repo and run

```
$ git pull
```

2 Running Code on EWS

Add the following lines to the end of your `~/.bashrc`:

```
module load mpich2
module load cuda-toolkit
module load intel-license/ews
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/class/cs232/REU/tbb/lib/
```

Then either restart your terminal or run

```
$ source ~/.bashrc
```

Failure to do this can result in one or all of the following:

- The timing library yelling at you and refusing to compile
- MPI yelling at you and either refusing to compile or refusing to run
- CUDA *not* yelling at you and running with no complaints, but *none* of your code works as intended

3 Problems

1. (30 points) **Getting Started**

   In this part of the MP, you'll familiarize yourself with OpenMP, MPI, and CUDA by writing some simple parallel code. Specifically, you'll write code to change all text in a file to upper-case. You can find the serial version of this code in `toupper_serial.c`.

   You can run this code with

```
./toupper_serial big_text_file.txt
```

The interface for each parallel version of `toUpper` is the same as the interface for the serial one.

You are provided with `shakespeare.txt`, which is a text file containing the complete text of every work of William Shakespeare. You also have access to `eight_shakespeares.txt`, which is the contents of `shakespeare.txt` repeated eight times. Both of these are good text files to use when timing your parallel code against the serial version.

Note that this code will overwrite the file you pass as an argument. It would be a good idea to make a copy of each `txt` file and run your code on that copy so you maintain the original version of the `txt` file.

(a) (10 points) Implement `toUpper` using OpenMP in `toupper_omp.c`.

(b) (10 points) Implement `toUpper` using MPI in `toupper_mpi.c`.

(c) (10 points) Implement `toUpper` using CUDA in `toupper_cuda.cu`.

2. (70 points) **Parallel Histograms**

In this part of the MP, you'll write code to count the frequency of characters in a text file. You'll start by writing serial code for this, and then move on to each of OpenMP, MPI, and CUDA.

You can run the serial version of this code with

```
./histogram_serial big_text_file.txt
```

If you want to see the transformed text, run

```
./histogram_serial -print big_text_file.txt
```

The interface for each parallel version of `histogram` is the same as the interface for the serial one.

(a) (5 points) Implement the serial version of `histogram` in `histogram_serial.c`.

(b) (15 points) Implement `histogram` using OpenMP in `histogram_omp.c`.

(c) (25 points) Implement `histogram` using MPI in `histogram_mpi.c`.

(d) (25 points) Implement `histogram` using CUDA in `histogram_cuda.cu`.

4 Turning in the Assignment

You will need to push all your solutions to your GitHub repo. To push `some_file.c`, do the following:

```
$ git add some_file.c
$ git commit -m "Informative commit message"
$ git push -u origin master
```

If you have any issues submitting your code, please make a post on Piazza and we will address the issue.

5 Grading

Your grade will depend on both your code's correctness and its running time. That is, your code should not only be correct, but it should be reasonably faster than the provided serial solution. Your code will be tested on EWS machines; you should do the same when evaluating your code's running time. In particular, do not use a remote connection to evaluate running time, as you may get worse results than expected.

DO NOT MODIFY THE PROVIDED MAKEFILE! If bugs are discovered in the Makefile, an announcement will be made on Piazza and a new Makefile will be pushed to your repos. Your code for each problem should be solely contained the corresponding file for that problem.

6 Beyond HW 2

If you want more practice with OpenMP, MPI, and CUDA, read on! None of this is required for the homework assignment, but working through these problems will give you invaluable experience working with these technologies. This is especially encouraged for anyone who plans to use one (or more!) of these technologies as part of their final project.

In these problems, you will implement Conway's Game of Life using OpenMP, MPI, and CUDA. If you are not familiar with the rules of Conway's Game of Life, you can read about it at `en.wikipedia.org/wiki/Conway's_Game_of_Life`.

A working implementation of Conway's Game of Life is provided to you in `conwaylife.c`. In this code, the board starts with the Die hard pattern, which vanishes after 130 iterations. If you implement your parallel Game of Life correctly, the Die hard will still vanish after 130 iterations. If not, something is wrong!

In particular, doing this in MPI is a good challenge for the serious MPI programmer. The communication pattern is very subtle; how will you split up the board? What kind of inter-process communication do you need?