This lab is meant to acquaint you with:

1. A C++ program meant to be multithread executed on a hyper-threading enabled computer
2. Timing the run-time of a command-line invoked program

Recall the following two diagrams from the Concurrency lecture, where threading was briefly explained:

The left diagram depicts an architecture where threading is disabled, hence instructions for threads u and v are processed on distinct pipelines. Black boxes represent stalls due to possible data or structural hazards. The diagram on the right depicts an architecture where threading is enabled; instructions for threads u and v are processed by the same pipeline. Threading (or multi-threading as it is often called), frees up pipelines for doing additional work, which potentially can increase overall CPU performance.

Although we’ll talk about threads in more detail in future weeks, this lab will introduce you to some of the threading constructs available to you. This lab will also demonstrate how different threads execute concurrently – it is not possible to predict the execution order by looking at the code alone.

As you did for the first lab, there are questions/tasks throughout these instructions that you should answer or follow and which you’ll submit to Canvas as proof of having completed the lab.

I. A simple thread program, scheduling

C and C++ provide constructs to write multi-threaded programs. Recall that two or more programs, or threads, run concurrently if you cannot tell from the code alone in which order they will be executed.

It takes very little to write a C++ program with threads.

- You must include the thread functionality with an #include directive at the top
- The keyword thread is used to refer to an instance of a thread object
As a first step, reproduce the code in the figure on the right in a file called threeThreads.cpp.

That program:

- Includes the <thread> functionality
- Declares a function, task, that takes as a single argument a reference variable to an object of type string. That argument is then echoed back via a cout command, which is the “print” command for C++
- Includes a main routine that creates three threads. Do not worry about what .join means. That will eventually be discussed in lecture when we talk at more depth about threads.

To compile the file threeThreads.cpp, you must explicitly specify that threading be used. To do so, issue the following compile command:

```
g++ threeThreads.cpp -o threeThreads -std=c++0x -pthread
```

To run the just-created executable threeThreads, issue the following command:

```./threeThreads```

Q1 : Run the program threeThreads 10 times in fast succession. Are the threads executed in the same sequence each time? Is there any odd behavior that you notice?

In addition to visually inspecting the output of a program, you can time a program to see how many seconds of processor and overall time it consumed. Issue the following command:

```
time ./../threeThreads
```

Notice three “outputs” provided by the time command: real, user, and sys. Look these up to find out what they mean.
Task 1: Take screen shots of the real, user, and sys outputs for one of the invocation of your program. Include the screenshot in your document.

II. A non-thread program, scheduling

Now that you have written a very basic threaded program, for this section of the lab you’ll write its non-threaded equivalent and use the time routine to compute its runtime and compare it to the threaded version of the program.

The non-threaded version of the threeThreads.cpp program is shown on the right. Re-create that in the file noThreads.cpp.

Notice that the threaded functionality is not included as a header, and the method task is invoked three times, but without the use of threads; the methods are invoked sequentially.

To compile noThreads.cpp, issue the following command:

```
g++ noThreads.cpp -o noThreads
```

As you did for threeThreads, run noThreads 10 times, and use the time command.

Q2: What is the execution order the three tasks for noThreads for your 10 invocations?

Q3: What is the approximate run-time of the noThreads program compared to the threeThreads program? If there is not much run-time difference between the two, then explain why? If there is a noticeable difference, then explain what advantage threaded or non-threading has over the other approach.
III. A more computationally intensive threaded program

The added advantage of threading is difficult to see in a program that is not computationally intensive. A thread must be computationally taxing enough so that the architecture is pushed past capacity before a difference between a threaded and equivalent non-threaded program can be seen.

For this portion of the lab, create two programs, `threaded.cpp` and `nonThreaded.cpp`. For them:

- Create a task function that is computationally intensive. For example, use a for loop with a large iteration count (start with 100,000 for example), and in the body of the for loop perform non-trivial floating point, ratio, or division calculations.
- Consider increasing the number of threads and/or concurrent invocations to task from 3 to 5, 10, or more.
- Just as you did for the previous sections of this lab, `threaded.cpp` and `nonThreaded.cpp` should be identical except that one uses threads and the other does not. Both should have the same count of threads/invocations, and the task method for both programs should also be the same.

Task 2: Cut and past the text of the computationally intensive calculation from the body of the task function into your writeup document.

Compile both programs.

Task 3: Systematically, run each program 10 times and record their run-times. If the run-times of the `threaded.cpp` and `nonThreaded.cpp` programs are similar, continue to modify the task function and make it more computationally demanding until there is a difference between the programs. Select which of the “time” outputs to use.

Task 4: Using the tallied run times from the previous step, generate a plot of the run-times for both the threaded and non-threaded versions of the program. Use the plotting program of your choice.

Be sure to:

- Give your plot a title
- Label the x and y axes
- Specify clearly which data series is threaded, and with is non-threaded

A sample plot is shown on the right. Note that the plot is for demonstration purposes; the values are not real, so your plot might look very different.
IV. Submission

Upload to Canvas the following:

- A document (.doc, .docx, .pdf) with your answers to questions and/or screenshots of completed tasks. **Be sure that your plot is included in your document.**
- The program `threaded.cpp`
- The program `nonThreaded.cpp`

V. Rubric

<table>
<thead>
<tr>
<th>Component of Lab</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tasks 1-4, questions 1-3</td>
<td>70 points</td>
</tr>
<tr>
<td><code>threaded.cpp</code></td>
<td>15 points</td>
</tr>
<tr>
<td><code>nonThreaded.cpp</code></td>
<td>15 points</td>
</tr>
</tbody>
</table>