EE 361C/382C: Multicore Computing

Instructor: Prof. Vijay Garg (email: garg@ece.utexas.edu)
TA: Miao Qi (email: qimiao@utexas.edu;)
TA: Asad Malik (email: asadsm@utexas.edu)

Assignment 2: Fall 2016
Deadline: Sept 27

This assignment has a programming component. The source code of the programming part must be submitted to Canvas before the end of the due date (i.e., 11:59pm **Sept 26**). Please zip and name the source code as EID1-EID2.zip. For each of the programming questions, you need to submit the source files. The nonprogramming part should be written or typed on a paper and submitted at the beginning of the class. The assignment should be done in teams of two.

1. **(10 points)** Consider the following problem called *Renaming*. There are n processes which have indices in range $1..N$ where $N >> n$. We would like these processes to get a new name in the range of $1..M$ where $M = n * (n + 1)/2$. How will you construct a function `int rename(int j)` that returns a unique name to each process in the range $1..M$. Every invocation of `rename()` should terminate and no two processes should obtain the same name. Your function should not use old index to *compute* a new index, i.e., it should satisfy the *index independence property* that if a process whose index is i obtains the new name v, then the process could have obtained the same new name v if its index had been j different from i. (Hint: Use enough splitters to split the input stream of threads).

2. **(30 points)** Write a program that uses n threads, where $n = 1, 2, 4, 8$. These threads increment a shared variable c. The total number of increment operations are $m = 1,200,000$. Each thread reads the value of c and increments it m/n times. Implement the following methods and compare the the total time taken for each of the following methods. Submit the plot as part of the assignment.
(a) Lamport's Fast Mutex Algorithm.
(b) Bakery Algorithm.
Note: Use arrays of Atomic Integers in your solutions

3. **(30 points)** Give an implementation of Java class java.util.concurrent.CyclicBarrier using semaphores. A CyclicBarrier is a synchronization aid that allows a set of threads to all wait for each other to reach a common barrier point. CyclicBarriers are useful in programs involving a fixed sized party of threads that must occasionally wait for each other. The barrier is called cyclic because it can be re-used after the waiting threads are released. You need to implement the following methods:

```
public CyclicBarrier(int parties) {
  // Creates a new CyclicBarrier that will trip when
  // the given number of parties (threads) are waiting upon it

  int await() throws InterruptedException {
    // Waits until all parties have invoked await on this barrier.
    // If the current thread is not the last to arrive then it is
    // disabled for thread scheduling purposes and lies dormant until
    // the last thread arrives.
    // Returns: the arrival index of the current thread, where index
    // (parties - 1) indicates the first to arrive and zero indicates
    // the last to arrive.
  }
}
```

4. **(30 points)** The purpose of this question is to learn OpenMP. To setup the API, you can install gcc-4.7 compiler on the Linux machine or Visual Studio 2008–2010 C++ on the Windows machine. For more information, please visit: `http://openmp.org/`. Alternatively, you can use TACC account for this question.

   (a) Write a C/C++ program `MatrixMult` that allows parallel multiplication for two matrices of doubles by using OpenMP. The task for your program is described below:

   Your program should accept three arguments. The first two arguments are paths of the input files that encode two matrices that need to be multiplied. The format of an input file is defined as follows: each input file contains one matrix. The first line of an input file contains two positive integers: m and n denoting the number of rows and columns in the matrix. The next m lines in the file provide rows of the matrices with entries separated by space. The third argument to your program, T, indicates the number of threads to be used. Suppose your program is named `run`, and is executed with the following parameters:

   `./run mfile1 mfile2 T`

The output of your program (to standard output) should be a matrix in the format used for input matrices. Assume that matrices are of the proper form and can be multiplied. Submit a plot of time taken to compute the product of matrices of size 100 by 100 when the number of threads are varied from 1 to 8.

(b) Write a multithreaded C/C++ function `MonteCarloPi` that calculates the value of π using the Monte Carlo Method. The function returns the value of π as a `double` and its signature is given as follows:

```
double MonteCarloPi(int s)
```

Imagine that we have a square with side length $2R$ and it contains a circle of radius R (see Figure 1). The idea of Monte Carlo Method is that if you randomly choose s points (black and grey dots) inside the square, there are c points (black dots) that are located in the circle, where $c/s = \pi/4$. In your implementation, you can choose your own value of R and use the equation, $x^2 + y^2 < R^2$, to check if the chosen points are located inside the circle. Your function should choose points in parallel.

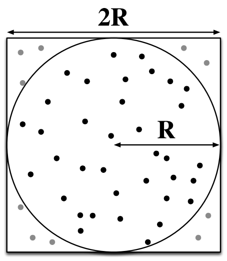

Figure 1: Black and grey dots are the randomly chosen points.