CSCI 176 (Parallel Processing) Spring 2016 Prog. Assignment 3 (25 pts) Due: March 29 (T)

Parallel matrix multiplication in Java Threads with shared-memory communication and monitor synchronization.

The matrix multiplication problem described in Prog2 is used again in this assignment, i.e.,

A(L*m) * B(m*n) = C(L*n)

where, $C_{i,j} = \sum_{k=0}^{m-1} (A_{i,k} * B_{j,k})$

For the implementation, we need the following 4 classes:

P3_matrix: the main class, which creates and run multiple threads to solve the problem in parallel;
Note that shared objects should be created and passed to each thread (as parameter to constructor).

Mythread: for multiple threads, each of which is responsible for partial computation.

Globals: for shared object, storing matrixes A, B, and C.

Communication: for shared object, which is responsible for communication/synchronization.

In the main process, access the number of threads to be used and matrix dimensions (L, m, n), and initialize the values of A and B matrixes with the following formula:

$A_{i,j} = i+j$; $B_{i,j} = i+j+1$;

After initializing A and B matrices, the main process should create the requested number of threads and let them work for building C matrix. The suggested method of dividing the work is using the start and end row numbers for the resulting matrix C, i.e., divide the first dimension (L) of matrix C with the number of threads. In the case that L is not evenly divisible by the number of threads, use the method of assigning one extra entry to some beginning threads.

Please use the following dimensions of the matrices for the final output to submit:
L=1001, m=2000, n=3000
Since the size of the output is extremely big (~36MB), please display only the first_20*first_10 and the last_20*last_10 entries of matrix C.

Run your program with number of threads = 1, 2 , 4, 8 and check/show the execution time for each run. For this, you need to use time checking codes in your program.

Submission:

Include good documentations and submit the hard copy of the source code and the run time output (from 4 runs).

Sample output format for num_threads=4:

```
L=1001, m=2000, n=3000
Thread_0: start_row=0, end_row=250
Thread_1: start_row=251, end_row=500
Thread_2: start_row=501, end_row=750
Thread_3: start_row=751, end_row=1000
=== C:  first_20*first_10 ===
. . . .
. . . .
=== C: last_20*last_10 ===
. . . .
. . . .
total computation time (sec) = 35
```

```
//////////////////////////////////////////////////////////////////////////////////
//// matrix multiplication with Java Threads – frame code
//// $>javac P3_frame.java
//// $>java P3_frame
//////////////////////////////////////////////////////////////////////////////////

public class P3_frame //main class; name must be same as the file name (P3.java);
                                    //in the single-file version, only main class is public;
{
  //// create a shared_object in class global;
  //// create a shared_object in class communication;

  public static void main(String args[])
  {
    //// matrix A,B,C memory allocation here;
    //// matrix A,B initialization here;

   for (int k=0; k<4; k++)
   { mythread temp = new mythread(k, glob, comm); //pass thread_id & shared_objs
      temp.start();
   }
  }//main
}

class globals //shared_obj for keeping matrix A,B,C
{
   public int[][] A; //A matrix declare
   public int[][] B; //B matrix declare
   public int[][] C; //C matrix declare
}

class communication //shared_obj for communication/synchronization
{ private static int counter = 0;

  public synchronized void increment(int id) //mutex
  { counter++;

     //// if Thread_1
     ////    check for wait() condition
     //// else
     ////    check for notify() condition
  }
}

class mythread extends Thread
{
   static globals glob;              //local for shared_obj
   static communication comm; //local for shared_obj
   private int id;

   public mythread(int k, globals glob, communication comm) //constructor
  {
     //// assign parameters to locals here;
  }

   public void run()
   {
     //// here matrix computation
     //// start/end index computation based on id

     //// increment done_counter in shared_obj comm;
     //// when all threads are done, display resulting C matrix in Thread_1

   }//run

}//class mythread
```