Homework 5: due by 11:59pm on Wednesday, October 17, 2012 (Total: 100 points)

Instructor: Vivek Sarkar
Co-Instructor: Ran Libeskind-Hadas

All homeworks should be submitted using the submission system at http://cs.hmc.edu/submit. In case of problems using the system, please email your homework to cs181ehelp@cs.hmc.edu. Note that this homework does *not* have a programming component.

Honor Code Policy: All submitted homeworks are expected to be the result of your individual effort. You are free to discuss course material and approaches to problems with your other classmates, the teaching assistants and the professor, but you should never misrepresent someone else's work as your own. If you use any material from external sources, you must provide proper attribution.

1 Object-Based Isolation and Linearizability (25 points)

Please submit your solution to this assignment in a plain text file named `hw5_1.txt` *in the submission system.*

1.1 Conversion of compareAndSet() calls to object-based isolated statements (10 points)

Rewrite the two do-while loops in class IQueue below to use object-based isolated statements instead of the compareAndSet() calls in method enq() and deq(), thereby making explicit the meaning of the compareAnd-Set() calls. You may need to split statements and add local variables to do this rewrite. (See expansion of AtomicInteger operations in slide 28 of Lecture 7.)

```
import java.util.concurrent.atomic.*;
 class IQueue {
   AtomicInteger head = new AtomicInteger(0);
   AtomicInteger tail = new AtomicInteger(0);
   Object[] items = new Object[Integer.MAX_VALUE];
   public void enq(Object x) {
     int slot ;
     do slot = tail.get(); // Loop till enqueue slot is found
     while (!tail.compareAndSet(slot,slot +1));
     items[slot] = x;
   } // enq()
   public Object deq() throws EmptyException {
     Object value; int slot;
     do { // Loop till dequeue slot is found
       slot = head.get(); value = items[slot];
       if (value == null) throw new EmptyException();
     } while (!head.compareAndSet(slot,slot+1));
     return value;
   } // deq()
 } // Iqueue
```

1.2 Linearizability Analysis (15 points)

Analyze the linearizability of instances of class IQueue as though they were concurrent objects. Either prove that all executions with calls to enq() and deq() will be linearizable, or provide a sample execution that is not linearizable. You can use either the original AtomicInteger version or the object-based isolated version in your solution to Problem 1.1 as the input program for your linearizability analysis.

2 Locality with Places and Distributions (25 points)

Please submit your solution to this assignment in a plain text file named `hw5_2.txt` *in the submission system.*

The use of the HJ place construct is motivated by improving locality in a computer system's memory hierarchy. We will use a very simple model of locality in this problem by focusing our attention on remote reads. A remote read is a read access on variable V performed by task T0 executing in place P0, such that the value in V read by T0 was written by another task T1 executing in place P1 $\neq$ P0. All other reads are local reads. By this definition, the read of A[0] in line 11 in the example code below is a local read and the read of A[1] in line 12 is a remote read, assuming this HJ program is run with the -places 2:1 option

```
1.        finish {
2.          place p0 = place.factory.place(0); place p1 = place.factory.place(1);
3.          double[] A = new double[2];
4.          finish {
5.            async at(p0) { A[0] = ... ; } async at(p1) { A[1] = ... ; }
6.          }
7.          async at(p0) {
8.            ... = A[0]; // Local read
9.            ... = A[1]; // Remote read
10.         }
11.       }
```

Consider the following variant of the one-dimensional iterative averaging example studied in the lectures. We are only concerned with local vs. remote reads in this example, and not with the overheads of creating `async` tasks.

```
1.      dist d = dist.factory.block([1:N]);
2.      for (point [iter] : [0:M-1]) {
3.        finish for(int j=1; j<=N; j++)
4.          async at(d[j]) {
5.            myNew[j] = (myVal[j-1] + myVal[j+1]) / 2.0;
6.          } //finish-for-async-at
7.        double[] temp = myNew; myNew = myVal; myVal = temp;
8.      } // for
```

1. **(10 points)** Estimate the total number of remote reads in this code as a symbolic function of the array size parameter, N, the number of iterations, M, and the number of places P (assuming that the HJ program was executed using the "-places P:1" option).
2. **(10 points)** Repeat part 1 above if line 1 was changed to dist d = dist.factory.cyclic([1:N]);
3. **(5 points)** What conclusions can you draw about the relative impact of block vs. cyclic distributions on the number of remote reads in this example?

3 Load Imbalance with Places and Distributions (25 points)

Please submit your solution to this assignment in a plain text file named `hw5_3.txt` *in the submission system.*

Consider the example code below that also uses places and distributions. In this example, we are only concerned with estimating the total number of operations performed at each place by adding up the contributions from calls to perf.addLocalOps() in line 6, as is done in HJ's abstract performance metrics.

1. **(10 points)** Estimate the total number of operations performed at place q ($0 <= q < P$) as a symbolic function of N, P, and q. For example, if P=1, then the total number of operations performed at place q=0 must be N*(N+1)/2.
2. **(10 points)** Repeat part 1 above if line 1 was changed to dist d = dist.factory.cyclic([1:N]);
3. **(5 points)** What conclusions can you draw about the relative impact of block vs. cyclic distributions in improving the load balance in this example?

```
1.        dist d = dist.factory.block([1:N]);
2.        finish for(int j=1; j<=N; j++)
3.           async at(d[j]) {
4.             for (int i=1; i<=j; i++) {
5.               ...
6.               perf.addLocalOps(1)
7.             }
8.        } //finish-for-async-at
```

4 Message Passing Interface (25 points)

Please submit your solution to this assignment in a plain text file named `hw5_4.txt` *in the submission system.*

Consider the MPI code fragment shown below when executed with two processes:

1. **(10 points)** What value will be output by the print statement in process 0?
2. **(15 points)** How will the output change if the Irecv() call is replaced by Recv() (and the Wait() call eliminated)?

```
int rank, size, next, prev;
int n1[] = new int[1]; int n2[] = new int[1];
int tag1 = 201, tag2 = 202;
Request request; Status status;

size = MPI.COMM_WORLD.Size();
rank = MPI.COMM_WORLD.Rank();
next = (rank + 1) % size;
prev = (rank + size - 1) %size;
n1[0] = rank*10 + 1; n2[0] = rank*10 + 2;

if ( rank == 0 ) {
  request= MPI.COMM_WORLD.Irecv(n1,0,1,MPI_INT,prev,tag1);
  MPI.COMM_WORLD.Send(n2,0,1,MPI_INT,next,tag2);
  status = MPI.COMM_WORLD.Wait(request);
  System.out.println(Output =  + n1[0]);
}
else { // rank == 1
  MPI.COMM_WORLD.Recv(n1,0,1,MPI_INT,prev,tag2);
  n2[0] = n1[0];
  MPI.COMM_WORLD.Send(n2,0,1,MPI_INT,next, tag1);
}
```