Homework 6: due by 11:55pm on 4/25/14, penalty-free extension till 5/2/14 (Total: 100 points)

Instructor: Vivek Sarkar

All homeworks should be submitted in a directory named `hw_6` using the turn-in script. In case of problems using the script, you should email a zip file containing the directory to `comp322-staff@mailman.rice.edu` before the deadline. See course wiki for late submission penalties. There are no programming assignments in this homework. All code snippets in the code listings are pseudocode.

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 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 Locality with Places and Distributions (35 points)

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 8 in the example code below is a local read and the read of A[1] in line 9 is a remote read, assuming this HJ program is run with 2 places, each place with one worker thread.

```
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. **(15 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 P places, 1 worker thread per place).
2. **(10 points)** Repeat part 1 above if line 1 was changed to `dist d = dist.factory.cyclic([1:N]);`
3. **(10 points)** What conclusions can you draw about the relative impact of block vs. cyclic distributions on the number of remote reads in this example?

2 Load Imbalance with Places and Distributions (35 points)

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. **(15 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. **(10 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
```

3 Message Passing Interface (30 points)

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

1. **(15 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)?

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

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

11.     if ( rank == 0 ) {
12.       request= MPI.COMM_WORLD.Irecv(n1,0,1,MPI_INT,prev,tag1);
13.       MPI.COMM_WORLD.Send(n2,0,1,MPI_INT,next,tag2);
14.       status = MPI.COMM_WORLD.Wait(request);
15.       System.out.println("Output = " + n1[0]);
16.     } else { // rank == 1
17.       MPI.COMM_WORLD.Recv(n1,0,1,MPI_INT,prev,tag2);
18.       n2[0] = n1[0];
19.       MPI.COMM_WORLD.Send(n2,0,1,MPI_INT,next, tag1);
20.     }
```