CS 320/CSE 302/ECE 392

Introduction to Parallel Programming for Scientists and Engineers
MP6 - Due Wednesday, May 6

The objective of this MP is to write an MPI Monte Carlo Dynamical Simulation in which idealized fish and sharks live, move randomly, breed, and eat one another in a two dimensional periodic ocean. The name WaTor derives from the toroidal topology of the imaginary watery planet (see enclosed article from Scientific American. December 1984).

The underlying space of the simulation is taken to be a two dimensional ocean with periodic boundary conditions of a torus. The domain is discrete in that fish and sharks are allowed to live only on points of a mesh.

The simulation input data consist of five numbers: *nfish* and *nshark*, the numbers of fish and sharks at the start of the simulation; *fbreed* and *sbreed*, the ages at which they breed; and finally, *starve*, the length of time a shark can survive without eating a fish. Here time refers to the number of iterations the simulation has undergone, so that the simulation in temporally as well as spatially discrete.

The fish move in a simple way. Each fish examines its four nearest-neighbor locations. If empty location are found, the fish is moved into one of these sites at random (using a `rand()` routine). If no empty locations are discovered, the fish remains where it is. In either case, the age of the fish is incremented. In addition, if the age has reached *fbreed* and the fish has actually moved, a new fish of age 0 is placed at the old location.

Sharks are similar to fish, except they are hungry (presumably, the fish are eating universally abundant plankton). This means that sharks look for fish at their four neighboring positions. If they are found, the shark picks one at random, eats it, and resets its starve parameter to zero. If no fish are nearby, the shark resorts to moving at random, just as the fish do. Again, the shark breeds if its age has reached *sbreed*. If a shark goes for starve time units without eating, it dies.

The program should be based on a few two dimensional arrays: *ocean(i,j), age(i,j),* and *starve(i,j)*. A fish at location 17,26 could be represented as a “1” in *ocean(17,26)*, a shark by a “2”, an empty spot by “0”. Notice that, because of the toroidal topology, if *ocean* has dimension *n* by *m*, the left neighbor of *ocean (i,1)* is *ocean(i,m)* and the upper neighbor of *ocean(1,i)* is *ocean(m,i)*.

Ocean should be divided among processors in a way similar to the rectangular domain decomposition used for the Poisson problem. The update step would then simply require a doubly nested set of loops which run over all locations (plus the necessary communication operations). To avoid collisions, you can think of the rectangular decomposition as a checkerboard where adjacent squares have different colors, say black and white. Then, computation would alternate between black and white blocks so that first all white block are updated. Then all black blocks. And so on. (This MP was extracted almost verbatim from: G. Fox et al. *Solving problems on concurrent processors*. Prentice Hall).

For the extra 50 points, a second version of the above program should be written. In this second version we will avoid having to go throughout all the ocean (where most locations could be empty) and instead associate several values with each point in the ocean array so that the program will know where in ocean is the next fish or shark given that you are at the location where a fish or sahrk is. It is recommended that you use four arrays: *next_x, next_y, previous_x*, and *previous_y* corresponding to the two coodinates of the next and the previous fish or shark. If there is no next fish the coordinates should be -1. Similarly for the previous pointer.

Code should be in your home directory (dated no later than May 6) under names mp5-1.f (the base problem) and mp5-2.f (the optional problem)