Exercise: The Game of Life

Thanks to John Horton Conway and British Mathematics

Westley: I mean, if we only had a wheelbarrow, that would be something.
Inigo Montoya: Where we did we put that wheelbarrow the albino had?
Fezzik: Over the albino, I think.
Westley: Well, why didn't you list that among our assets in the first place?

What are our Assets?

1. We have the experience of doing the area under the curve and working with the n-body code
2. We have the Game of Life code that Dave Joiner wrote and Fitz further scaffolded
3. You have an idiot and a fool helping guide you
   (we entering at messy confusing process that will hopefully be successfully scaffolded)
4. We are setting aside worrying about display (though that is the whole reason for the Game of Life)
5. Googling "Game of Life" gives us boatloads more examples and descriptions

What is our Goal?

1. Analyze/Design/Code a problem slightly more complicated than Area Under a Curve
2. Explore the parallel pattern that can be solved by maintaining a shared boundary
3. Explore MPI_Sendrecv, instead of carefully structured independent MPI_Send and MPI_Receive

How we Going to Get from Here to There?

1. Lets make sure we understand the Game of Life problem
   a. In a 2D grid, a square has 8 neighbors
   b. In Game of Life, each generation only depends on the previous generation
   c. Rules for making a generation are simple
      i. An empty square (dead) with exactly 3 live neighbors, will live in the next generation
      ii. A filled square (alive) with 2 or 3 neighbors will survive in the next generation
      iii. All other alive squares die in the next generation
2. Lets design a serial version of the code where
   a. We tie the top of the grid to the bottom of the grid
   b. We tie the left side of the grid to the right side of the grid
   c. Make a display function (that will eventually be X-windows) to display the grid
   d. We parameterize the size of the grid, so we can print the grid when the size is small
3. Lets talk about ways of dividing the task and the prices paid
4. Lets talk about ways of exchanging boundaries between two processes
5. Lets actually change the code and make it run in parallel using MPI

Cellular Automata
The Game of Life

The Game of Life is an iterative process set up on a square grid. Cells on the grid are either “alive” or “dead”. If a cell is “dead” and has exactly 3 neighbors, it has enough resources to be born without being overcrowded, and the next turn will be “alive” If a cell is alive and has 3 or 4 neighbors, it has resources without being overcrowded and will stay “alive”. If a cell has 2 or fewer neighbors, it cannot get enough resources to survive and the next turn will be “dead”. If a cell has more than 4 neighbors, it will be overcrowded and the next turn will be dead.

A typical progression might look like:

Turn 1

			X
	X		
	X		
	X		

Turn 2

X	X	X	

Turn 3

	X		
	X		
	X		

These simple rules can lead to many complicated phenomena, some of which seem quite stable, and some of which seem almost chaotic.

Running the Game of Life on a large scale can require a lot of memory. The amount of storage scales as the side length of your grid squared.

This problem is ripe for exploitation by parallel programming. You could break up a larger grid into smaller subgrids. Since each cell only needs information about its nearest neighbors, you only have to communicate among subgrids at the edges of the subgrids.

The MPI Life example is set up to run as “side by side” subgrids. You enter in the number of rows and columns of each subgrid, and the number of iterations to be solved.

Try the following

```
time mpirun –np    2       Life       100          50          1000             1
                 #cpus              #nrows    #ncols    #niterations    do_display
```

Compare it to

```
time mpirun –np 1 Life 100 100 1000 1
```

Does using more CPUs allow you to solve the same problem faster?

What is the efficiency of this implementation on your cluster?

#cpus	Real time	efficiency
1		XXXXXXXXXXXXXXXX
2		
3		

(efficiency can be measured in many ways, but typically can be expressed by taking the running time with 1 processor, and dividing it by the running time with P processors *P)
efficiency = time(1)/(time(P)*P)

What is meant by the efficiency of a parallel solution? If you calculate it once, doe sit apply to any parallel code running on that machine?