Parallel Design Patterns

Assessed Coursework, 2016
PART TWO

The deadline for PART TWO of the assessed coursework is Friday 1st of April at 12 noon. You will submit this via the PDP learn pages.

About the coursework

Part two of the coursework follows on from your initial work on part one. However, these two pieces are marked independently and your grade for part two will not depend on the answers given in part one.

In this part of the coursework you will:

- Write parallel code which implements the Biologist's model using an **Actor pattern [75%]**
- Write a report that explains your implementation **[25%]**

Your code should be written in Fortran, C or C++ and parallelised with MPI.

It is possible to receive a very good mark by writing a code that follows the pattern correctly. An excellent solution (worthy of distinction level marks) should split the code into two parts:

1. **A framework** implementing the actor pattern but contains no problem specific code
2. **Problem specific code** which uses the framework to solve the biologist's problem.

Your code should compile and run on Morar. You are advised to use the Portland Group compilers with MPICH2, but you may use the GNU compiler and/or OpenMPI if you state that you are doing this in a README file submitted with your source code.

As described in the lectures, an actor pattern can be implemented by either one actor per process or multiple actors per process. You may choose which ever approach you find easiest, I would expect that the first one is easier to code. An excellent solution can be based on either solution.

The asynchrony that the actor pattern promotes, along with the fact that the biologist's provided functions step squirrels to random grid locations, will result in non-determinism between exact results from one run to the next. Instead scientists would run your model multiple times and take averages of the results to get a final answer. This is fine and there is no need to worry about exact reproducibility, either from run to run or machines to machine (with different hardware capabilities.)

It is acknowledged that the implementations of MPI on Morar have their limitations, in particular in terms of the number of concurrent processes that can run. Do not let this concern you for the assessment! If you need to, just limit the number of squirrels that you are simulating so that your code will run. Do not feel that you must work around this limitation by using multiple actors per process if you do not wish to write your code in this way.

Code [75%]

Your code should:

- Compile and run on Morar.
- Correctly implement the biologist's model.
- Be clear and adopt a clean, ideally modular, design.

- Be packaged neatly with a **README** file describing how the code should be built and run. Ideally a **makefile** and example submission script will also be included.
- Be adequately commented to a level that would allow others to work on your code in the future.

Performance is only a secondary consideration in this assessment. Clarity, flexibility and correctness are more important.

Output

For each simulated month your simulation should output for each land cell:

- The population influx
- The infection level

An excellent solution would also output, for each month, the total number of currently alive squirrels and the number that are infected.

Suggested Parameters

The following parameters will give you an idea of the scale of simulations that the biologists wish to perform, but these will not influence your choice of pattern.

- Initial number of squirrels: 34
- Number of cells: 16
- Maximum allowed number of squirrels: 200
    - The biologists don't think the habitat can ever home more than this number, if the number exceeds 200 then your simulation should terminate with an error message
- Initial infection level: 4 squirrels at the start of a simulation
- Months to model: 24 although it should be possible to easily modify this

Report [25%]

Your report should mainly focus on the design of your implementation, and explain how the pattern has been applied to the problem. If you have written a separate framework then you should briefly document how it is designed to be used.

As stated above, performance is only a secondary consideration in this assignment, but credit will be given in the report for any discussion about aspects of the code's design that help or hinder performance.

Your report should show the results obtained by running your code, this is mainly so that the marker can see you have successfully run the code. In the marks for the report you are being graded mainly on how well you describe your implementation, and the clarity with which this is presented. A discussion of the output of the simulation can gain you some extra credit, but lengthy descriptions or explanations about the output are not required to gain a very good mark.

Code provided to you

You will be provided with

1. A set of functions that the biologists have used in previous serial codes. Your parallel code should use the functions provided.
2. A basic code that provides an implementation of a process pool. It is entirely your choice whether you use this or not, but students often find it useful to have this provided. This was used as part of the fifth practical (mergesort) and there is a test driver to illustrate using this, there is no need to understand the internals of the process pool code if you do not wish to – you can use it by treating it as a black box.

The Biologist's Functions

C is used in the descriptions below, but an equivalent set of Fortran functions have also been provided in the code package which is available on Learn for download. Note that in the Fortran versions, the *logical* type is used as the return type for functions where appropriate.

- void **initialiseRNG**(long* seed)
  - Call this once at the start of your program **on each process**. The value should be **negative**, **non-zero** and **different on every process**. You could seed with the value *-1-rank* where *rank* is the process's MPI rank. Out output, *seed*, becomes the random number generator's *state* that is passed to other functions.
- void **squirrelStep**(float x, float y, float* x_new, float* y_new, long* state)
  - Returns the position of a squirrel after it has stepped from position (*x*,*y*) to a new position, (*x_new, y_new*.) *x_new* and *y_new* can point to *x* and *y*.
- int **willGiveBirth**(float avg_pop, long* state)
  - Given the average population of the cells visited (i.e. the sum of their *populationInflux* values divided by the number of steps this has been taken over, 50), this returns true or false depicting whether the squirrel has given birth or not.
- int **willCatchDisease**(float avg_inf_level, long* state)
  - Given the average inflection level of the cells visited (i.e. the sum of their *infectionLevel* values divided by the number of steps that this has been taken over, 50), this returns true or false depicting whether the squirrel has caught the disease.
- int **willDie**(long* state)
  - Returns true with a probability of 1 in 6. This is called for a squirrel to see if it is to die when it is infected (it should do 50 infected steps before calling this.)
- int **getCellFromPosition**(float x, float y)
  - Given an (*x*,*y*) co-ordinate pair, this returns an integer representing the land cell that this corresponds to.

Details of the biologist's model

This is the same as explained in the assignment for part one

The model that the biologists have written exhibits the following behaviour:

- Squirrels can move about their environment
- A squirrel's current position in the environment is described by a coordinate pair (x,y)
- A squirrel moves by stepping to a new position
- Squirrels can be born
  - In this model all squirrels give birth to new squirrels
  - New squirrels start in the same position as their parents
  - A squirrel can move as soon as it is born. We assume all new squirrels are adults, capable as any other squirrel (we ignore the early phases in a squirrel's life.)
  - All squirrels are born healthy (but see note about the initial infection at the start of the simulation.)
- Squirrels can catch squirrel parapoxvirus
  - The animals either have the disease or not (so their state of infection can be represented as a boolean.)
  - A squirrel never recovers from the disease, they have it until they die
- Squirrels can die; they are normally long lived so in this model they only die from being infected by the disease.

- The environment in which the squirrels live is modelled as a set of grid cells. In their current model, this is a regular grid, but they would like the parallel framework to be able to deal with more general grids in the future. The biologists have supplied a function which converts an (x,y) position to the grid cell (an integer cell number).
- Time is modelled coarsely, in terms of months
    - Squirrels can take many steps in a single month
    - Squirrels move throughout the entire month, they have no concept of global time or seasons.
    - The squirrel parapoxvirus virus can live without a host in the environment for 2 months.
- The disease is passed through the environment
- Land cells each have a populationInflux value calculated as the total number of squirrels that have stepped into the specific cell at some point during the past 3 months.
    - Squirrels that are born in a cell do not contribute to the populationInflux at the instant that they are born
    - If a squirrel steps and ends up in the same cell, this counts as a squirrel arriving in the cell. (In other words, every hop can be treated in the same way, as a squirrel arriving in a cell.)
    - Each time the month changes the cells should be updated to represent the past 3 months
- Land cells each have an *infectionLevel* which is calculated as the total number of infected squirrels that have been in the specific cell in the past 2 months.
    - All squirrels are born healthy so won't contribute to this level
    - If a squirrel steps and ends up in the same cell, this counts as a squirrel arriving in the cell. (In other words, every hop can be treated in the same way, as a squirrel arriving in a cell.)
    - Each time the month changes the cells should be updated to represent the past 2 months
- After every 50 steps, a squirrel will reproduce with a probability depending only on the average *populationInflux* of all of the cells they have visited in their last 50 steps.
- After every step, the squirrel will catch the disease with a probability depending on the *infectionLevel* of all the cells they have visited in the last 50 steps.
- An infected squirrel will live for a minimum of 50 steps, after this then they will die with a fixed probability of 1/6 for each step after.

Having Difficulty?

A code that does not quite work might be good enough to pass as long as the ideas are correct and the code is accompanied by a good quality report. In the event that a non-working code is submitted, the report should explain the parts that *do* work, should describe the symptoms of why the program is not working and the steps taken to try and fix the problems.

If you are struggling to implement all of the details of the biologist's model, then you may wish to solve a simpler case. If this is the case, you should state clearly in your report which simplifications you have made. A working code for a simplified model could gain as good a mark or better than a broken code what aims to implement the full model.

If all else fails, use the report to describe how you would have solved the problem given more time. Describe the code that you *have* submitted.