

# Parallel Design Patterns 1

B138813

## 1(a).

The first pattern we recommend for this problem is the **Actor Pattern**. We make this suggestion as the problem domain can be expressed as entities, which map, on a 1:1 basis, to actors. These actors are squirrels, grid cells and a simulation master. Squirrel and grid cell actors have a one to one mapping with the functionality described in the details of the biologists model, i.e. a grid cell will have a `populationInflux` value. This mapping is a notable advantage as it will make it much easier for biologists to understand the final code. The simulation master would be able to act as a clock, to instruct grid cells if a two or three month interval had passed, which would then adjust the `populationInflux` and `infectionLevel` of all land cells. The simulation master could also keep a list of live and dead squirrels, to ensure that no more than 200 squirrels are created, and to terminate the program if all squirrels die or the simulation time has elapsed. When a squirrel is born, the simulation master can simply 'resurrect' a squirrel processes to be reused.

Another key advantage of this model is its extensibility, in that if the biologists wish to create a predator animal for the simulation, they only need to create another actor, rather than restructuring and rethinking their entire code base. This pattern also gives us an opportunity to reuse the biologist's function to map the x,y location of a squirrel to a grid cell number, and make that grid cell number the rank of the grid process in an MPI implementation. This code reuse is desirable for two reasons, firstly it saves us development time, and secondly it actively involves the biologists in the development of their model, which should lead to better development outcomes.

However, that is not to say there are not drawbacks to the actor pattern for this problem domain. It is very difficult to design any form of locality in the actor pattern, which means that squirrel processes may communicate with grid processes very far away from themselves, leading to latency. For this model, the number of cells is 16 and the maximum number of squirrels is 200, so data locality is not expected to become a bottleneck, but it may do so at larger scales. It is also noteworthy that the actor pattern can be very difficult to debug, as squirrels and land cells will interact in a way that inevitably becomes harder and harder to determine as the simulation continues. Whilst this unpredictability does closely follow the biologist's model, and helps avoid bulk synchronicity emerging from the implementation, it will make the simulation difficult to debug.

We also recommend the use of **2D Geometric Decomposition** with halo swapping for this problem. We make this suggestion because we believe that a geometric decomposition

will allow us to exploit the local nature of the problem. Each grid square would be similar to a land cell, with all the values and functions as in the actor pattern. However, in the Geometric Decomposition we would bundle these functions along with other functions in a general `process()` function for each grid square. The `process()` function would also move squirrels between grid squares and manage squirrel mortality. Squirrels would be represented by data structures, which could be exchanged across UEs through halo swaps. In this scenario the simulation would perform a global reduction at monthly intervals, to collect statistics on squirrels and infection levels for the global simulation. These monthly reductions would also allow for the `infectionLevel` and `populationInflux` level to be updated at the same time across the global simulation.

Overall, a geometric decomposition has many advantages, but we remain sceptical of it's utility for two central reasons. Firstly, this design lacks robustness. The biologists would like some integral policy in the future, that is they would "like the parallel framework to deal with more general grids". In this geometric decomposition the user's policy (layout of grids) profoundly impacts the mechanism's implementation (halo swaps etc). The communications in a geometric decomposition are fundamentally tied to the the geometry of the problem, which means that the code would need significant rewriting to deal with different geometries. Secondly, too much code is bundled in the `process()` function. Even if it is fairly inexpensive to call this function, it violates the rule of modality, as everything is done by one function, and it violates the rule of transparency, hiding much functionality behind an opaque function.

## 1(b).

Ultimately, we recommend using the actor pattern for this problem. Although use of the actor pattern does imply a higher level of communication between UEs, it also prevents our solution from becoming too rigid. A 2D geometric decomposition can only easily scale between differently sized quadrilateral grids, and we must build a tool which can be adapted later for differently shaped grids, i.e. an irregular blob shape constructed from a pentagonal grid. The actor pattern will certainly require more UEs if each squirrel and each grid cell has its own UE, but this should not be too much of a problem on a machine like Archer. Fundamentally, it is the extensibility of the actor pattern which leads us to recommend it.

## 1(c).

We would implement the actor pattern with MPI and C. We have chosen MPI because the actor pattern does not require

shared state, and dictates that all parallelism must occur through communication. These principles are central to the use MPI, therefore using it will keep our implementation true to our design. We have also chosen to use C because whilst it is not a particularly memory safe language, it is one of the fastest programming languages available, with a larger pool of people able to develop in it than Fortran, which also has compatibility with MPI.

For hardware we would preferably use a large compute cluster of roughly 15 nodes, where each node in the cluster has the ability to run at least 16 parallel processes (non-GPU). We suggest this as it would give us more than enough processes to run 200 squirrels, 16 grid squares and 1 squirrel master, with room to begin testing larger grids. On Cirrus we would require only seven nodes.

### 2(a).

We recommend against using the **Pipeline Pattern**. It will be unsuitable no matter what data elements the problem is decomposed into, as a sense of global time will be needed at monthly intervals to assess the `populationInflux` and `infectionLevel` variables. As such, the pipeline will regularly block until all elements, squirrels or grids, are at the same time step. This stuttering of time spent processing would lead to bulk synchronicity, preventing operations from being run concurrently on different data elements. Further to this, the scenario requires that there is some sort of interaction between grid cells and squirrels, which this pattern does not accommodate for. It is also easier to optimise the pipeline pattern with a data set of a constant size. However, as the number of squirrels is expected to fluctuate, we will not be able to configure the correct depth for this simulation.

It is also conceptually difficult to map this problem to **Recursive Task Parallelism**. Whilst the squirrel simulation can be decomposed into tasks, it does not fully map to a recursive task pattern, which works best when tasks are completely asynchronous and independent. Unfortunately, as we found with the pipeline pattern, the simulation must synchronise each month to calculate the `populationInflux` and `infectionLevel` values, which would require tasks to interact with each other. However, even if this hurdle were overcome using some sort of shared memory or global reduction, either with openMP or PGAS, we would still face the problem of how to break this problem down into recursive tasks. We could potentially think of squirrels as tasks, moving across a grid held in shared memory, and spawning new squirrel tasks when appropriate. The major downside of this approach is that we would not have any way to track the global number of squirrels in the simulation, and be able to shut it down when we reach 200 squirrels.

### 3(a).

You would be able to use the actor pattern again for this problem, and have each grid cell actor communicate with its neighbours to spread the disease. This could easily be done by creating a 2d cartesian communicator for these messages

to spread between their neighbours, which could be found by performing `MPI_Cart_shift`. This would lead to a sort of simple 2d decomposition hybrid with the actor pattern, which differs from a true 2d decomposition from its lack of halo swaps. The tenants of the actor pattern, (that everything is an actor, that information is not stored in shared memory but is passed through messages) are still obeyed, except now that the disease is a factor of the land cells rather than the squirrels. This simple relocation of functionality implies that it should be easy enough to rewrite the actor pattern first proposed into this area.

However, if the biologists present a particularly complicated model of the disease spreading, it is recommended that we begin to manage the pathogens themselves as actors. We make this recommendation so that our implementation remains in line with the idea of the actor pattern, and to strive towards an approach that is loosely coupled whilst retaining high code cohesion. For example, if the biologists later want the disease to react to certain environmental variables propagated from the simulation master, it would be better for these messages to go to pathogen actors, who will be affected by them, rather than to the land cells, who will largely not be affected by them, except for any disease code unnecessarily bundled with them.