Task 1

Question a

Commencing with building a biological model, the focus of the design is on looking for the tradeoff of parallelizing independent computation and reducing parallelization overheads. Load balance and scalability should also be considered significantly. In this case, the **geometric decomposition pattern** and the **actor pattern** are applicable to this model.

The **geometric decomposition pattern** could contribute to conducting this simulation efficiently. By splitting up the whole grid into roughly equal sub-grids and assigning each sub-grid to a UE, the decomposition can be achieved in 2 dimensions and the computation on each sub-grid can be performed concurrently. Since cells are independent to each other, swapping halos are not necessary. Initially, some squirrels(or 0) are set in each sub-grid. When a squirrel steps out of this area, all information of this squirrel should be sent to the corresponding sub-grid, which means that it moves from this sub-grid to another one. If a squirrel comes to this UE, vice versa.

For each UE, there are two main parts of the computation needed to be considered. One is the update of each squirrel's trajectory and state, and the other is the computation of *populationInflux* and *infectionLevel* on each cell.

- When a squirrel steps to a cell, the cell's *populationflux* and *infectionLevel* should be added into corresponding arrays respecting to the squirrel's trajectory for the last 50 steps. Then, whether it reproduces, is infected or dies will be computed according to average *populationInflux*, *infectionLevel* and death probability.
- For cells, when a squirrel moves into a cell for once, the corresponding cell should record this action and its infection condition. Every month, the *populationInflux* and *infectionLevel* should be calculated and stored in the cell's information.

It is worth noting that a master UE should do the clock computation, which will send a signal to other UEs per 'month'. In this way, cells in each UE could update the *populationflux* and *infectionLevel* regularly.

The granularity of data decomposition is essential. Finding the tradeoff between the communication overheads and parallelization by doing experiments is the key. The model with too coarse-grained design will weaken the benefit brought by parallelization, however, one with too fine-grained design will ineluctably increase the communication overheads. Besides, if the data structure is not regular enough, the unbalanced workload in each UE will be more serious as the sub-grid gets smaller, thus impacting overall system resource utilization.

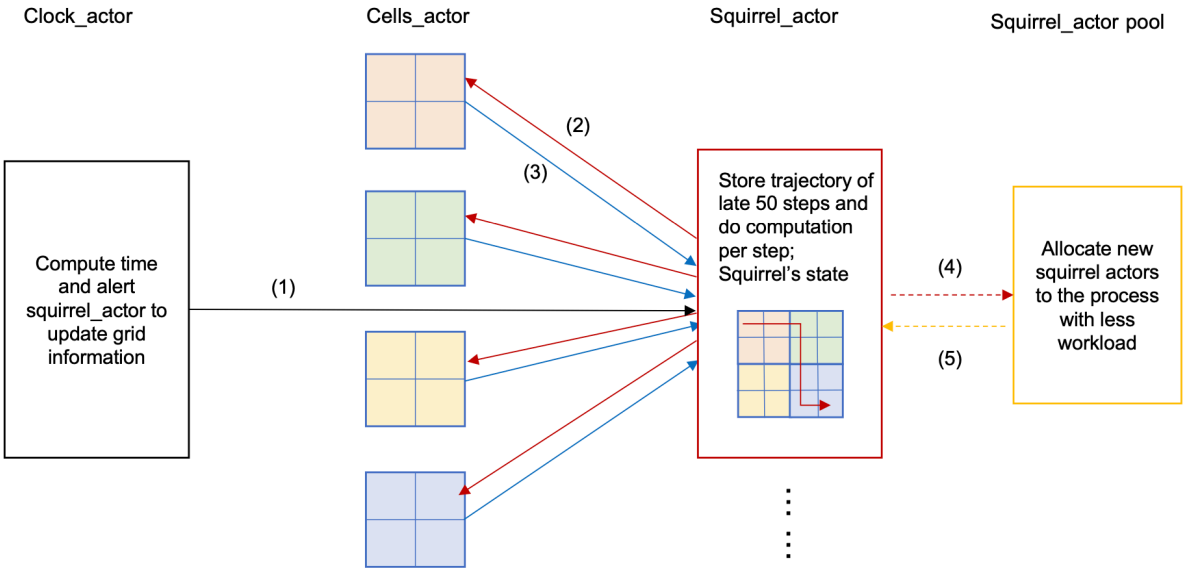
The main advantage of this pattern is that it can effectively reduce the overall execution time by parallelizing the squirrels' and cells' computation. When a squirrel moves around in the same sub-grid, it is easy to compute the data required for each step. When a squirrel moves out of this sub-grid, non-blocking communication can help in sending a 'squirrel' to another one. In addition, this pattern can be simply implemented by using MPI. It is easy for biologists to learn, debug, and apply this model into more complex cases.

However, it still has some limitations, especially in scalability. Biologists require that this model design should be able to deal with more general grids which can be irregular and large in the future. In this case, this pattern shows its weakness in keeping load balance for each UE and a lot of experiments of adjusting granularity are required to optimize the performance of this model.

The **actor pattern** could also be considered. For this model, every squirrel is an actor, and a group of cells is an actor, and a clock actor is required to manage the time of this model. Besides, a squirrel actor pool could be used to allocate new squirrel actors to different UEs. Figure a-1 shows the design

of this pattern.

- A squirrel_actor's computation only happens when the squirrel moves. In each squirrel_actor, it stores the information of the whole grid including *populationInflux* and *infectionLevel* for each cell. When the squirrel moves for a step to a cell, it will compute its state(infected, reproduce, die or not) based on its local cell information. In addition, the corresponding cell will record this action and its infected condition. When the squirrel_actor receives the message from the clock_actor, it will scatter the whole grid to each cell_actor. Next step, it will wait for cells_actors' messages and then combine the updated sub-grids.
- As for a cells_actor, it could be a slice of the whole grid or just a single cell. These cells_actors' work involves two parts: one is to receive each squirrel_actor's local grids and sum up the new month's population and infection number for each cell and then update *populationInflux* and *infectionLevel*; The other is to send back the new sub-grid to each squirrel_actors.
- The squirrel_actor pool is also an actor. It will constantly wait for the messages from squirrel_actors about the reproduction and death information, then take place of the dead squirrel_actor with the new one or allocate a new place for the new.



(1) The clock_actor will send a message to each squirrel_actor when the month changes.
(2) After a squirrel_actor receives clock_actor's message, it will scatter the whole grid to corresponding cells_actors.
(3) After a cells_actor receives all squirrel_actors' messages, it will do sum reduction and update its cells' information, then it will send back its updated sub-grid to each squirrel_actor. Each squirrel_actor will combine these received sub-grids into the updated new grid.
(4) If the squirrel_actor is dead or it reproduces a new squirrel, it will send a message to the squirrel_actor pool.
(5) The pool will allocate a new squirrel_actor to its place if a new squirrel is born.

Figure a-1

The actor pattern has many advantages. Firstly, the squirrel_actor pool could help in load balance to a large extent, thus improving the resource utilization of the system. Then, this pattern provides large flexibility for developers to modify this model according to different conditions. For example, if the number of UEs is limited, several squirrel_actors could be combined together and share the same grid by implementing multithreading. The size of cells_actors could also be smaller if the entire grid is big and irregular. In addition, it is easy to add new variables for different cases. It can simulate the complex nature well by parallelizing the changes of things as the way they happen in reality.

The disadvantage of this pattern is that it may cause heavy communication overheads due to many messages being delivered simultaneously, and the execution of this model is subject to the buffer of these messages, which needs to be fully considered. Another weakness is that when the whole grid is rather big, each squirrel_actor would store the exactly same grid information, resulting in

generating many replicates and wasting resources of the system. In addition, each cells_actor should be aware of the total number of squirrels and it may easily stall or conduct wrong results if a squirrel_actor's message is missing.

Question b

Compared to the geometric decomposition, the **actor pattern** would perform better. Since actors are self-contained and atomic, it gives developers more flexibility to modify this pattern based on different requirements, such as adding new types of actors. As for parallelization, the actor pattern could expose more parallelism. Squirrel_actors can work simultaneously just like how squirrels behave in the nature, and cells_actors are only active when they receive messages from aquirrel_actors. In addition, the actor pattern behaves better in dealing with an irregular grid than geometric decomposition. It takes less effort than the geometric decomposition pattern to modify the model – it could just change cells_actors' structure instead of restructuring the whole model.

Question c

In the actor pattern, each actor needs to send and receive messages. In this case, it is appropriate to call the MPI library to achieve frequent communications between actors. If there is a limited number of UEs, the OpenMP library can also be used in the squirrels_actor which computes several squirrels' behaviors and assigns each squirrel into a thread to achieve parallelization. Therefore, C, C++ and Fortran which are often used with the MPI and the OpenMP libraries are applicable to this model. For hardware platform, this pattern will perform well in clusters with many compute nodes, such as Cirrus and Archer. Because they could generate low communication overhead while delivering messages between nodes compared to the distributed architecture.

Task 2

The pipeline pattern and the recursive data pattern are not suitable for this model.

As for the pipeline pattern, it focuses on a number of operations working on each data element in a sequence. However, for this biological model, squirrels' next steps are randomly decided during simulation, and cells' update work is not continuous in the time dimension. A possible algorithm-level pipeline could be the process of updating the cell's current month's population and infection number when a squirrel steps into this cell since this process requires ordering constraints. Obviously, this pipeline needs only 2 or 3 stages, which is not efficient. Therefore, it is not appropriate to use the pipeline pattern for this case.

The recursive data pattern can be tough to apply for this model. Sometimes it could help an algorithm with recursive data structures such as a tree structure to execute by parallelism, but it still has a lot of limitations. It is difficult to find recursive structures in this model because its data structure is not sequential, rather, squirrels' next steps are randomly determined and cells' information changes accordingly. Therefore, this pattern is not suitable for this model.

Task 3

The geometric decomposition pattern is applicable to this case. This model needs interaction between neighboring cells and there is no inherent hierarchy in this data structure, therefore, it will be convenient for developers to implement 2-dimensional topology by calling MPI library. It could divide the entire grid into many sub-grids and each of them is allocated to a specific UE which maps to the corresponding location of the grid. Each UE could execute concurrently. In addition, they could swap halos with their neighboring UEs simply by naming them up_nbr, down_nbr, left_nbr and right_nbr to achieve data sharing. When applying this pattern to this model, the granularity of the data decomposition should be considered. It's important to find a tradeoff between the size of parallelized sub-grid and communication overheads to achieve the best performance.