|epcc|

The 2^{nd} Coursework Report for Parallel Design Pattern

B157041

April 9, 2020

Contents

1 Introduction

This report is for the second coursework of Parallel Design Pattern. The problem is a UK red squirrel disease simulation. There is a British red squirrels species suffering from a disease call *squirrel parapoxvirus* threatening the survival of the red squirrel. Biologists would like to save this species and work on try understand the spread of the disease crossing the population, and eventually protect them. Therefore, they would like to build a simulation model on HPC system and observe the behaviors of squirrels population and environment.

Our task is to write the parallel simulation by providing the serial code and process pool code from biologists and EPCC experts. This report is to demonstrate the detail of the implementation of the parallel simulation. Section 2 shows the architecture of the simulation including the actor's design and the framework we used. In section 3, the output of the simulation is displayed and correctness, as well as performance, is discussed.

2 Architecture

The parallel design pattern is **Actor Pattern** in this implementation. The participants of the simulation are **squirrels** and **land** (it might be called **cell** as well) which can be abstracted as actors. For better managing the simulation, we design a **controller** actor and **master** actor help to coordinate the communication between processes and get simulation output regularly.

The programming language we used here is C with pure MPI library. The experiment runs on Cirrus HPC server with double cores CPU (Intel® Xeon® Processor E5-2695 v4, 45Mb Cache, 2.10GHz). The OS is CentOS Linux 7 (Core). The compiler used to compile the source code is intel-compiler-18 and MPI library.

2.1 Actor model

The simulation can run concurrently. According to the coursework description, the behaviours of squirrels are moving, catching the disease, giving birth and die. The actions of land are calculating as well as updating population influx and infection level. By running the simulation multiprocessing, the squirrels' behaviours and lands' can be executed on different process concurrently, and there is message synchronisation if a squirrel steps on a cell of land. The squirrel's actions are dependent since whether catching the disease, giving birth and die need the result of moving; therefore, the activities inside a squirrel is not splittable. The behaviours between squirrels are irrelevant so they can execute their actions parallelly. The cells also can run concurrently as well without interference to each other except that they need to synchronise when the new month update. The controller is to monitor all the squirrels in simulation so it only need to receive squirrels' new states when their states change, and transmit the *updated message* to all cells to update the new month.

Therefore, we have the actor model of the simulation (in fact, the implementation would be a little different) as Figure ref fig: actor showing. We design that one actor running on one process according to the communication in the model. Our philosophy of the implementation is **trying to do many to many on the most frequent communication situation.** In every step, which is the most frequent situation, the squirrel and cell need to communicate, therefore, we arrange multi squirrel actors communicate with multi cell actors in order to disperse communication pressure. The number of controller actor can be adequately one since the communicate involving controller is not much frequent. It only occurs when the squirrel limitedly catches the disease, reproduce and die, and the cells' monthly synchronisation.

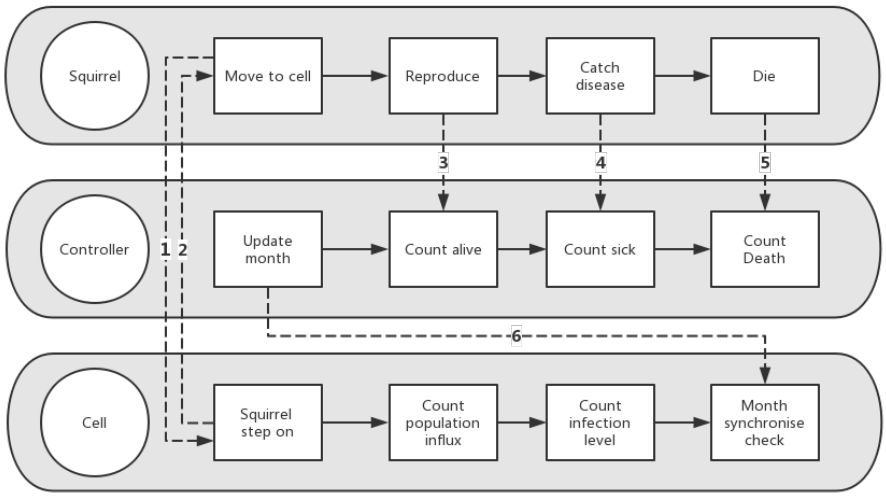

Figure 1: Concurrency Analysis. Each process inside a bar represent they need to run serially, and the different bar means they execute parallelly. The dotted arrow stands for the necessary communication. Communication 1 is the squirrel send its state to cell for counting population influx and infection level. Communication 2 is the cell send the last 3 months population influx and the previous 2 months infection level back. Communication 3, 4 and 5 is the squirrel transmit its state (alive, sick, dead) to the controller for monitoring all squirrels in the simulation. Communication 6 is the controller send the new month to all cells to update a new month synchronously.

2.2 Implementation

The implementation of the simulation basically follows the actor model described in section 2.1 with limited modification as Figure 2 showing. The figure is a little complex to read so We give the explanations of every stages below.

2.2.1 Squirrel walking

In the implementation, we allocate an individual actor for all cells of land. The squirrel arriving at a particular cell needs to communicate with the cell actor, to sending its state and receive the message. This message might be 2 integer data or 0 data and the squirrel take further action depending on the size of the message it got. If it gets 2 data, it means the message is the sum of population influx in the last 3 months and the sum of infection level in the previous 2 months of the cell it stepped on. The squirrel can try to reproduce (Figure 3), catch disease and die. The squirrel sends the its state to controller once their state change. For example, they report controller if their state becomes *sick* from *healthy*.

On the contrary, the squirrels are terminated and exit the loop if it get 0 data since we design that let the cell announce the squirrels to stop in order to stop the simulation. The stop mechanism of the simulation will be explained below.

2.2.2 Cells' response

We design that the cell will receive the message from squirrels or controller for different purposes. Therefore, we use *MPI_Probe()* to detect the source of message. (In fact, the *MPI_Iprobe()* is used in the code for increase the performance). If the message is from the squirrel, the cell should update the population influx and infection level send them back or send 0 data depending on whether the simulation is stop. The process 3 in Figure 2 define what message should be send back to squirrel.

The cells would know whether the simulation is stopped and the response to terminate the squirrel when it is stopped. If the message is from the controller, it might be the new month updated, the signal that let the cells terminate squirrels or the signal that the cells can stop. Process 4 in Figure 2 is the action the cells should take after receiving the message from the controller:

1. If the message is the new month, then the cells will send their population influx and infection level to the controller and reach the Barrier waiting for all cells to reach here.
2. If the message is the signal for stop squirrels, it will change the message on Process 3 and send 0 data to response squirrels so that the squirrels will be terminated.
3. If the message is the signal for stop cells, it will send the message to tell controller it has been stopped and quit the work on judgement 5.

The detail of interaction in process 3 and 4 can be seen in Figure 5. Here we should stop cells after squirrels stopped since it might cause deadlock of the entire simulation if the cells have been stopped as well as the squirrels still waiting for the response when arriving at this cell. The reason why we let the cells terminate the squirrel instead of making the controller stop them one by one is still the philosophy in section 2.1. We try to avoid many-to-one communication since it might cause to bottleneck.

2.2.3 Controller scheduling

The communications of the controller in the simulation are infrequent communications because it only occurs when the month update, the squirrels' state change. The controller is to monitoring the number of squirrels with different states and terminate the entire simulation when the simulation does not satisfy the condition. In this coursework, the simulation would be terminated by one of three situations.

1. The number of month is over the limitation.
2. The number of squirrels is more than the simulation allowed.
3. All the squirrels are dead.

The Controller treats a fixed period of actual execution time as the gap of a month. In every iteration of the loop, the controller would receive the changed state from squirrels and judge if there is a new month. When a new month arrived, the controller would send a new month signal to all cells and receive the population influx and infection level of every cells (Here we use non-blocking send and receive).

The controller stops the simulation if the condition is not satisfied any more. In Process 2 of Figure 2, the controller would send the Stop Squirrels signal to cells and waiting for the squirrels sending. When the cells receive that signal, they send 0 data to squirrels, and the squirrels' states would turn into *TERMINATE* and send this state to the controller. The controller receives the *TERMINATE* signal and knows a squirrel has been stopped. Once there are no active squirrel actors, the controller would send the Cell stop signal to all cells and shut down the pool, then the cells would stop, and the simulation is over.

2.2.4 Squirrel reproduce

The Process 1 in Figure 2 is the reproduce process of a squirrel. Due to the design of *one actor with one process*, the process of reproducing is to create a new process from the processes pool. Before squirrel giving birth, squirrels would enquiry the controller whether they can give birth or not since there is a limitation of the maximum number of squirrels in the simulation. The squirrels would give birth only when the controller allows.

2.3 Data structure

The crucial data structures of cells, squirrels and the controller are given on Figure 6. For each squirrels, we need their current coordinate which is x and y pair, the state identifying healthy or sick, the total number of steps and the number of steps after sick. We also need the last 50 steps of population influx and infection level for calculating the possibility of giving birth and catching disease. Meanwhile, the squirrels need to know who is controller and what pids of all cells.

The cell also need to know the controller's pid for communicating and the pids of other cells for generating group communicator in MPI. The cell would need to know the current month and only need to store the last 3 months of population influx and last 2 months infection level. The cells can store the population influx and infection level in the array by placing them into $month\%3$ and $month\%2$.

The controller should store the cells' pids and the current month for updating cells' month. It also need to store the number of squirrels with different states.

2.4 Code explanations

The code used for the coursework is based on the *test.c* file and modified by ourselves and the *pool.c* and *squirrel-functions.c* provided from EPCC and biologists are also used. This section is to help the code reviewers to read the project clearly.

2.4.1 Parameters

We put the parameters of the simulation into *include/config.h*. User can modify the parameters inside to configure the simulation. Here is the description of the parameters which can also be found on *readme.md* file.

- **CONTROLLER_NUMBER** is the number of controller actor. In this simulation, **we are against that the user set it more than 1**.
- **LENGTH_OF_LAND** is the number of land actors.
- **MONTH_LIMIT is** the number of land actors.
- **LAND_RENEW_RATE** is the time of a month that the land update the population influx and infection level.
- **LAST_POPULATION_MONTHS** Land update the population influx after this number of months.
- **LAST_INFECTION_MONTHS** Land update the infection level after this number of months.
- **MAX_SQUIRREL_NUMBER** is the maximum number of Squirrels. The controller terminate the simulation if the number of active squirrels is over this number.

- **INITIAL_NUMBER_OF_SQUIRRELS** is the number of initial squirrels.
- **INITIAL_INFECTION_LEVEL** is the number of initial sick squirrels.
- **LAST_POPULATION_STEPS** Squirrels are going to try give birth in every this number steps.
- **GIVE_BIRTH_STEPS** Squirrels try to give birth in every this number steps.
- **CATCH_DISEASE_STEPS** Squirrels try to catch disease after this number steps.
- **CATCH_DISEASE_STEPS** Squirrels try to catch disease after this number steps.
- **LAST_POPULATION_STEPS** Squirrels try to give birth according to the average of this number steps population influx.
- **LAST_INFECTION_STEPS** Squirrels try to catch disease according to the average of this number steps infection level.

2.4.2 Framework

The code is based on the following framework. The main function is in the *framework.c* and the structure is Listing 1. The *statusCode* 1 is the worker code and the pre-define actor identity is in *include/actorConfig.h* file. We should only add the new case inside the switch clause and call the *workerCode()* by inputting corresponding initialise function's and worker function's pointers implementing in actor themselves.

Listing 1 The main function in framework

```
int statusCode = processPoolInit();

if (statusCode == 1) {
    int identity;
    MPI_Recv(&identity, 1, MPI_INT, MPI_ANY_SOURCE, IDENTITY_TAG, MPI_COMM_WORLD, MPI_STATUS_

    switch (identity){
        case WORKER1:
            workerCode(initialiseWorker1, Worker1Code);
            break;
        case WORKER2:
            workerCode(initialiseWorker2, Worker2Code);
            break;
        ...
    }

} else if (statusCode == 2) {
    masterCode();
}

processPoolFinalise();
```

The *masterCode()* follow the structure in Listing 1. At first, the master would initialise the global variables that can be access by other actors' framework functions, then the master should call the *masterInitialiseWorkers(WORKER1, n1, worker1Pids)* and *masterSendWorkers(n1, worker1Pids, worker1AskFunc)* for each actors with their identity, the number of them and their Ask Function pointer implementing in actor themselves. The *worker1Pids* is the array to contain the Process Id of the brunch of worker 1.

Listing 2 The structure of master code

```
masterInitialiseVariables();  // Initialise the variables

masterInitialiseWorkers(WORKER1, n1, worker1Pids);  // Initialise the workers 1
masterInitialiseWorkers(WORKER2, n2, worker2Pids);  // Initialise the workers 2
...

masterSendWorkers(n1, worker1Pids, worker1AskFunc);  // Response workers 1 ask
masterSendWorkers(n2, worker2Pids, worker2AskFunc);  // Response workers 2 ask
...

int masterStatus = masterPoll();
while (masterStatus) {
    masterStatus=masterPoll();
}
```

The functions of an actor are following the actor framework structure. The squirrel actor is an example showing in Listing 2. The functions of an actor can be divided into 2 part. The first part is the framework functions that are accessible for master. It includes 3 functions, the Ask Function, the Initialise Function and the Worker Function, and these 3 functions will be called in *framework.c*. The Ask Function is for the actor to require the message from the master (the message typically is the other actors pids). The Initialise Function is for the actor to receive the message from the master corresponding with the Ask Function. The Worker Function is the main work of this actor and it would be called in the framework's switch clause.

The second part of the functions is private belonging to this actor and they define exclusively the actions of this actor would take.

Listing 3 The structure of actor functions

```
/** The functions blow from actor framework, they will be called in framework.c **/
int squirrelAsk(int workerPid);
int initialiseSquirrel();
int squirrelWorker();

/** The functions blow belong to this actor **/
int squirlGo();
void reproduce();
float get_avg_inf_level();
float get_avg_pop();
```

3 Outputs

Given the parameters in the code, we run the code on the backend of Cirrus with 7 computing nodes, 218 cores, and the stable outputs as Table 1 showing. The outputs include the number of alive and infected squirrels in the specific months and the number of total dead squirrels in history. The squirrels with different state in every month is showing on Figure **??**. From the number of alive, infected and dead changing, we can see the number of alive and infected are dynamically balanced and the number of all dead are growing. We think the result makes sense.

Month 1	ALIVE	33	CELL	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	INFECTED	2	POPULATION INFLUX	156	159	166	156	181	177	168	151	155	202	164	158	167	171	161	160
	DEAD	5	INFECTION LEVEL	23	18	16	18	16	25	27	18	25	35	21	28	22	18	30	17
Month 2	ALIVE	34	CELL	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	INFECTED	2	POPULATION INFLUX	92	80	81	78	90	77	84	82	91	92	80	80	84	171	79	93
	DEAD	9	INFECTION LEVEL	8	7	10	9	11	15	18	15	15	18	20	16	10	18	13	17
Month 3	ALIVE	32	CELL	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	INFECTED	0	POPULATION INFLUX	71	79	82	72	70	61	68	78	78	72	80	54	86	66	69	56
	DEAD	13	INFECTION LEVEL	10	9	10	9	6	8	11	6	13	8	9	6	17	8	11	7
Month 4	ALIVE	37	CELL	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	INFECTED	5	POPULATION INFLUX	68	63	57	60	74	83	60	48	55	54	48	67	51	65	66	63
	DEAD	14	INFECTION LEVEL	19	16	12	17	18	21	18	11	15	16	10	17	13	18	14	22
Month 5	ALIVE	37	CELL	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	INFECTED	4	POPULATION INFLUX	25	19	16	14	20	17	21	20	24	16	16	22	21	22	16	18
	DEAD	18	INFECTION LEVEL	8	4	7	7	4	5	5	4	8	8	4	8	7	6	7	4
Month 6	ALIVE	38	CELL	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	INFECTED	2	POPULATION INFLUX	41	55	56	63	67	51	68	48	56	60	51	67	56	66	62	53
	DEAD	22	INFECTION LEVEL	6	19	15	15	16	16	24	7	10	12	13	16	14	20	14	11
Month 7	ALIVE	41	CELL	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	INFECTED	4	POPULATION INFLUX	59	61	78	66	72	60	67	80	58	69	80	67	77	68	71	73
	DEAD	24	INFECTION LEVEL	6	5	11	6	8	11	3	12	6	10	11	16	8	13	14	8
Month 8	ALIVE	42	CELL	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	INFECTED	3	POPULATION INFLUX	92	93	71	81	96	111	85	100	100	90	93	90	83	86	74	84
	DEAD	28	INFECTION LEVEL	12	20	17	15	14	18	10	16	20	22	20	12	19	8	17	19
Month 9	ALIVE	41	CELL	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	INFECTED	4	POPULATION INFLUX	41	64	59	52	53	50	51	45	51	54	54	55	54	48	48	64
	DEAD	32	INFECTION LEVEL	15	22	14	12	16	14	11	12	13	14	13	18	14	8	17	19
Month 10	ALIVE	35	CELL	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	INFECTED	2	POPULATION INFLUX	62	62	76	60	66	65	57	61	68	60	51	60	43	51	70	56
	DEAD	39	INFECTION LEVEL	16	18	20	17	16	13	15	19	19	17	15	18	10	11	25	18
Month 11	ALIVE	37	CELL	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	INFECTED	3	POPULATION INFLUX	56	54	51	65	46	46	62	58	39	55	55	62	57	41	45	51
	DEAD	42	INFECTION LEVEL	12	17	14	21	19	10	18	15	9	20	15	17	21	7	10	19
Month 12	ALIVE	39	CELL	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	INFECTED	3	POPULATION INFLUX	43	45	41	41	49	56	49	34	46	46	40	38	53	51	46	41
	DEAD	45	INFECTION LEVEL	10	9	7	6	13	12	15	12	10	11	8	7	16	12	8	10
Month 13	ALIVE	40	CELL	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	INFECTED	4	POPULATION INFLUX	38	37	49	44	59	48	57	37	42	45	59	49	41	35	48	42
	DEAD	48	INFECTION LEVEL	9	11	9	6	9	7	13	5	9	11	10	11	9	5	3	7
Month 14	ALIVE	44	CELL	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	INFECTED	6	POPULATION INFLUX	29	41	35	56	38	42	44	42	41	41	46	45	37	38	52	45
	DEAD	49	INFECTION LEVEL	7	14	13	15	12	11	15	13	11	17	13	16	15	17	20	20
Month 15	ALIVE	40	CELL	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	INFECTED	2	POPULATION INFLUX	95	91	92	97	90	99	89	102	83	81	99	72	90	84	101	93
	DEAD	56	INFECTION LEVEL	18	23	17	20	15	19	23	18	15	17	16	21	19	19	17	12
Month 16	ALIVE	43	CELL	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	INFECTED	4	POPULATION INFLUX	26	37	35	36	39	31	31	36	45	46	35	34	36	34	32	49
	DEAD	58	INFECTION LEVEL	10	14	13	8	17	8	6	9	18	14	11	7	13	13	9	17
Month 17	ALIVE	41	CELL	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	INFECTED	1	POPULATION INFLUX	115	123	114	94	103	116	94	106	106	109	93	102	81	101	108	104
	DEAD	64	INFECTION LEVEL	10	14	18	10	15	18	15	10	8	9	12	8	7	11	9	11
Month 18	ALIVE	44	CELL	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	INFECTED	3	POPULATION INFLUX	72	86	78	83	84	75	91	106	86	78	84	92	82	87	100	84
	DEAD	66	INFECTION LEVEL	9	15	12	13	14	18	19	15	17	17	15	17	7	19	17	12
Month 19	ALIVE	42	CELL	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	INFECTED	5	POPULATION INFLUX	50	46	67	51	35	57	67	40	62	51	64	57	49	53	47	61
	DEAD	70	INFECTION LEVEL	15	13	18	17	6	23	18	13	11	17	11	19	13	11	11	21
Month 20	ALIVE	42	CELL	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	INFECTED	7	POPULATION INFLUX	41	33	36	39	33	30	47	38	41	33	35	46	53	50	25	43
	DEAD	73	INFECTION LEVEL	20	19	13	13	17	22	24	10	14	10	15	15	24	23	7	19
Month 21	ALIVE	40	CELL	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	INFECTED	1	POPULATION INFLUX	44	33	35	42	39	43	42	41	33	53	36	46	40	51	33	36
	DEAD	79	INFECTION LEVEL	10	11	9	9	12	10	10	8	9	16	8	15	9	15	5	10
Month 22	ALIVE	43	CELL	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	INFECTED	1	POPULATION INFLUX	61	55	61	57	70	45	55	63	49	61	59	57	65	52	71	62
	DEAD	82	INFECTION LEVEL	3	7	12	10	16	8	10	9	12	8	9	6	15	6	13	9
Month 23	ALIVE	46	CELL	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	INFECTED	3	POPULATION INFLUX	57	75	70	68	86	83	83	82	84	77	85	85	90	73	71	72
	DEAD	84	INFECTION LEVEL	6	7	7	4	4	9	9	11	9	11	9	10	9	8	6	12
Month 24	ALIVE	49	CELL	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	INFECTED	4	POPULATION INFLUX	60	68	58	61	59	49	51	56	71	41	51	61	65	54	59	67
	DEAD	86	INFECTION LEVEL	18	10	20	12	11	13	14	14	25	13	14	19	16	15	10	19

Table 1: The outputs of simulation

3.1 Correctness

Due to the unpredictable of output is one of the feature of Actor pattern, we would like to check the reasonability of the outputs according to the logic of simulation. Here we try to estimate the average steps of healthy squirrels and sick squirrels to see if they are similar. We calculate the approximate average steps by equation 1 since the approximate number of actives squirrels in this month is the number of alive and dead in this month minus the dead last month.

$$AvgSteps = \frac{total\_steps}{this\_month\_alive + this\_month\_dead - last\_month\_dead} \tag{1}$$

The approximate sick steps is calculated by equation 2. Therefore, we will see the changing of average of total steps and sick steps are similar if the outputs are logically correct.

$$AvgSickStep = \frac{total\_sicksteps}{this\_month\_sick + this\_month\_dead + last\_month\_sick - last\_month\_dead} \tag{2}$$

We draw the line plot of the average steps estimation in Figure 8 to visualise the trend. From the curves showing, we can see the changing of two kinds of steps are basically similar, which means our simulation is correct.

3.2 Performance

This section list the points about what we done in the code for help to increase the performance and what hinder the performance. The outputs include the runtime logged in master process and we conclude that the runtime is less than 0.05s when the simulation running on the backend of Cirrus.

3.2.1 Help performance

From the design perspective, we follow the philosophy which is **trying to do many to many on the most frequent communication situation.** In fact, we do multi-squirrels and multi-cells model to balance the workload since we meet the bottleneck if we make many squirrel actors communicate with one land actors frequently. In the terminating stage, we design that the controller let multi-cells to terminate multi-squirrels in order to prevent the bottleneck of one controller terminating all actors.

From the implementation perspective, we try to use MPI standard send much as we can to improve the robust of the simulation since the standard send initially do the non-blocking buffer send, which follow the "fire and forgot" feature of Actor Pattern, and do the blocking synchronous send only when the buffer size is not enough. We use *MPI_Iprobe()* in cells actor as well for non-blocking probing.

3.2.2 Hinder performance

The shortage of hindering performance is the blocking receive, *MPI_Recv()*, using in the simulation. We decide to use *MPI_Recv()* to guarantee the correctness of the simulation. Still, the blocking receive might cause the number of the unstable steps since the interval of the month is counting on actual time including the communication time when the processes are blocking. The outputs table 1 shows that the number of total steps in the fifth month is relatively low because of this.

4 Conclusion

In conclusion, the report shows the complete development process of the red squirrel simulation by using the actor pattern, including the model design, code description and output analysis. We design the process of parallel running for each actor and the terminate mechanism of the simulation and implementing the design by C with pure MPI. After testing, the simulation is resilient and robust, and would not meet deadlock or crash. We analyse the output from the number of squirrels with different state and their average steps in each month and can conclude that the outputs are logically correct. Finally, we give a simple performance analysis for this simulation. Further work can work on performance improvement.

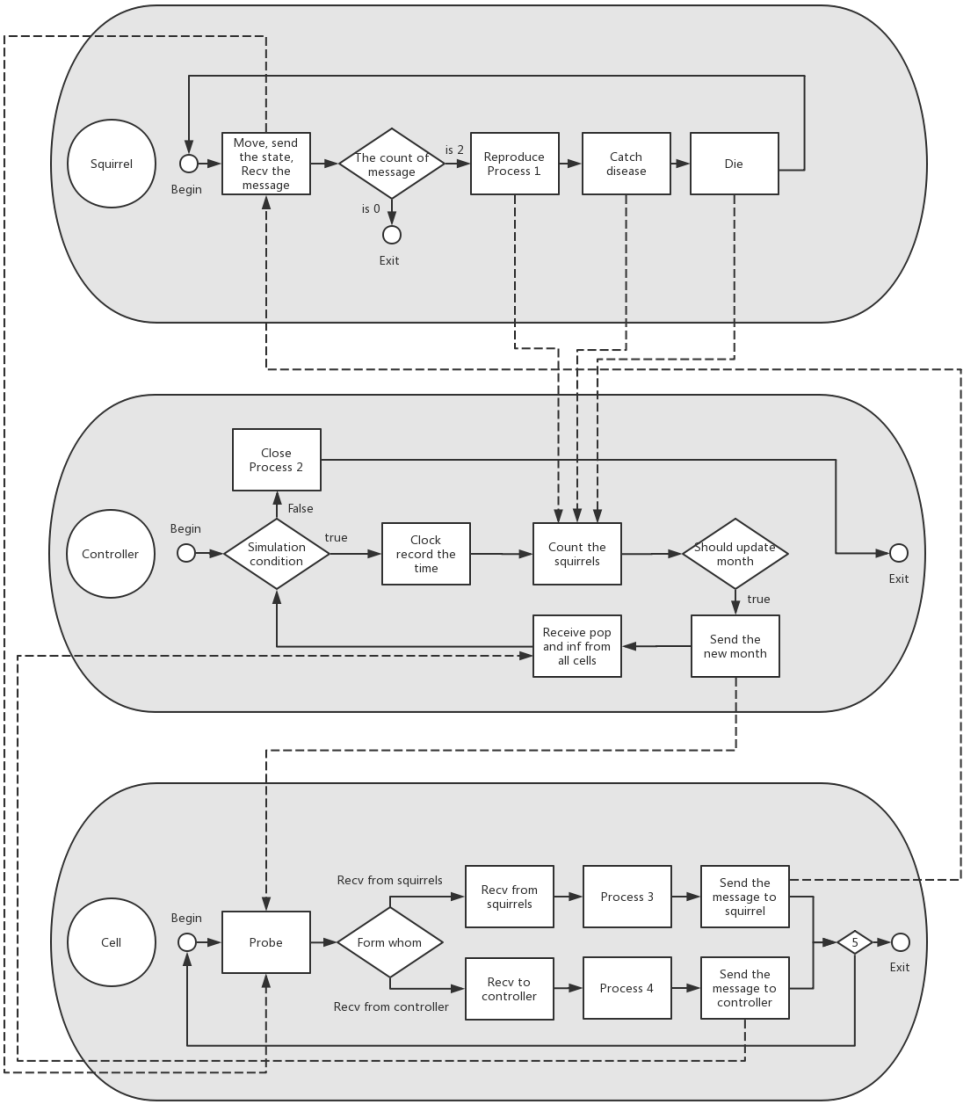

Figure 2: The parallel simulation flow chart. The solid arrows are the flow direction and dotted arrows represent communication. This is the normal stepping flow chart. The process 1-4 and a judgement 5 are for termination of the simulation and are explained in other figures.

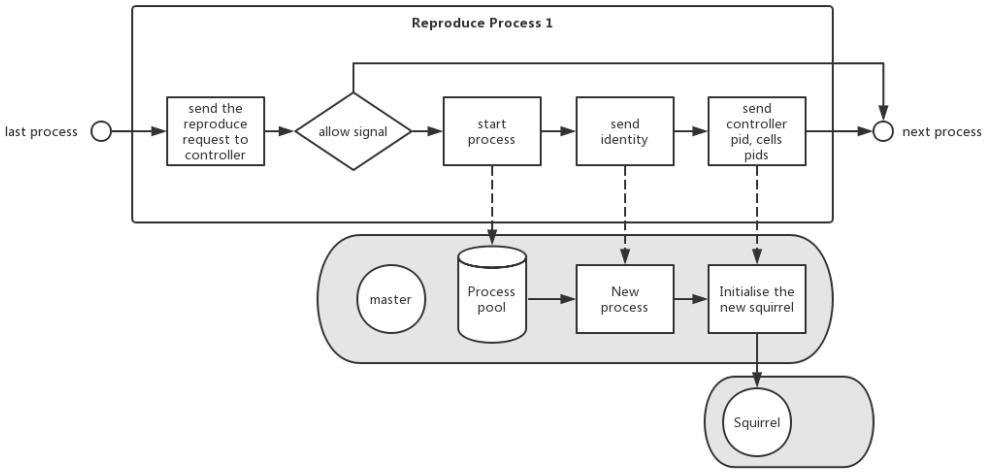

Figure 3: The inner detail of process 1.

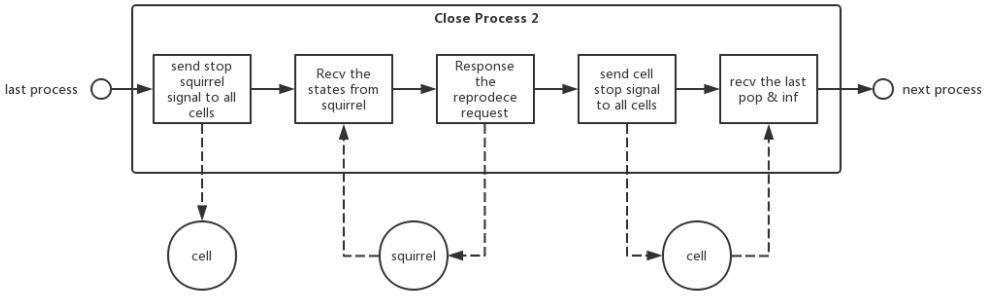

Figure 4: The inner detail of process 2.

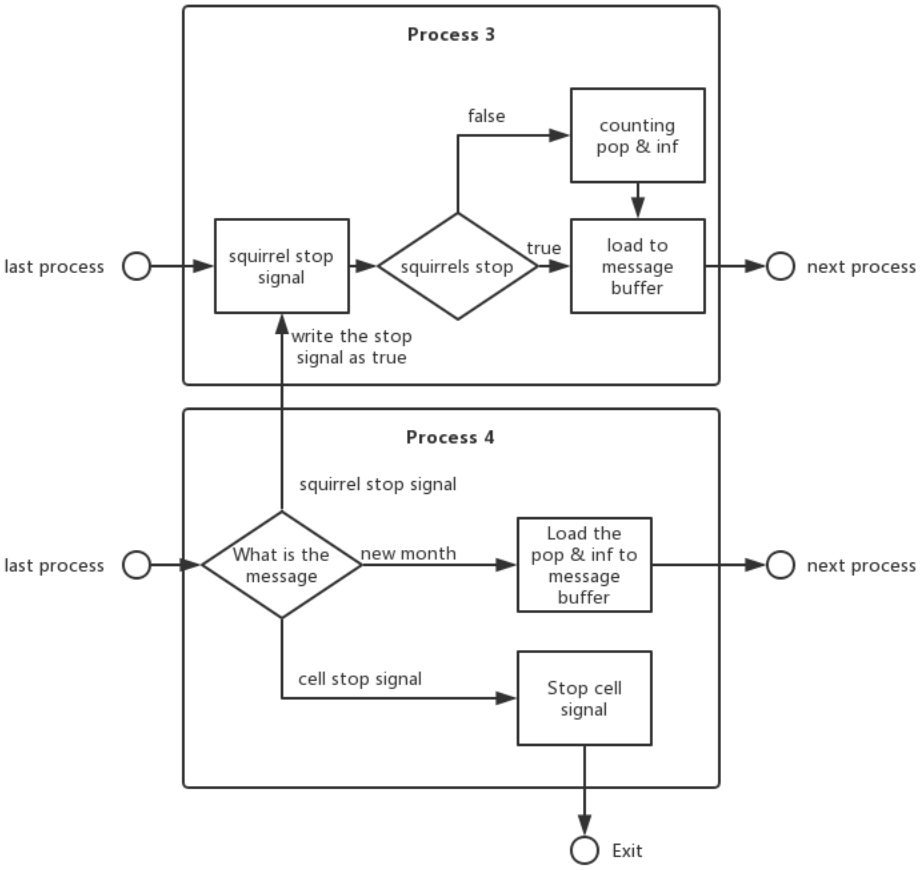

Figure 5: The inner detail of process 3 and process 4.

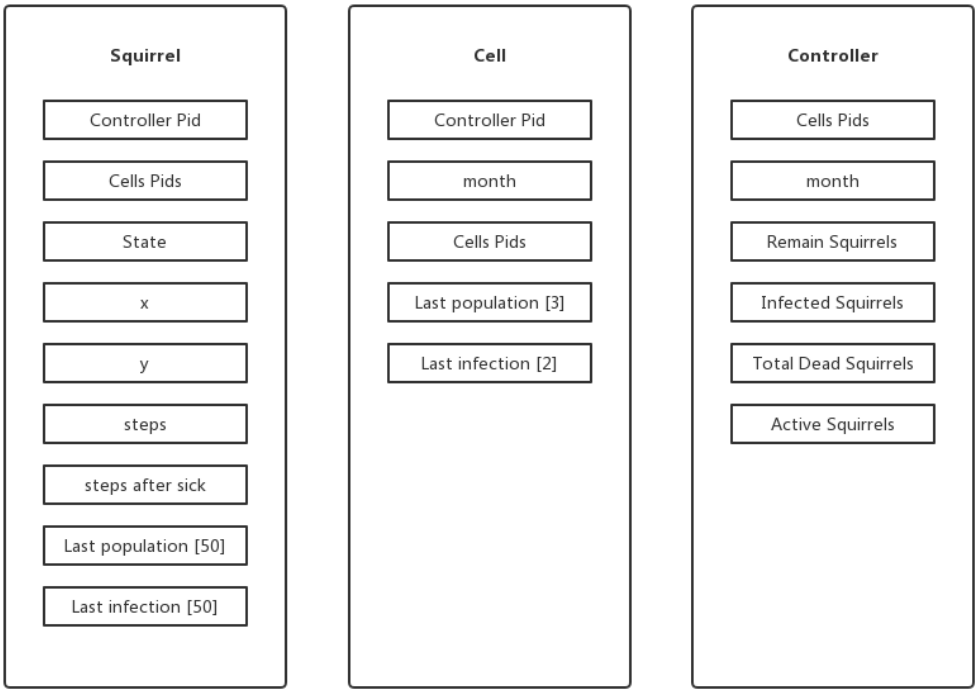

Figure 6: The data structure of each actor. The type of all the variables or array of variables is integer except x and y. The type of x and y is float.

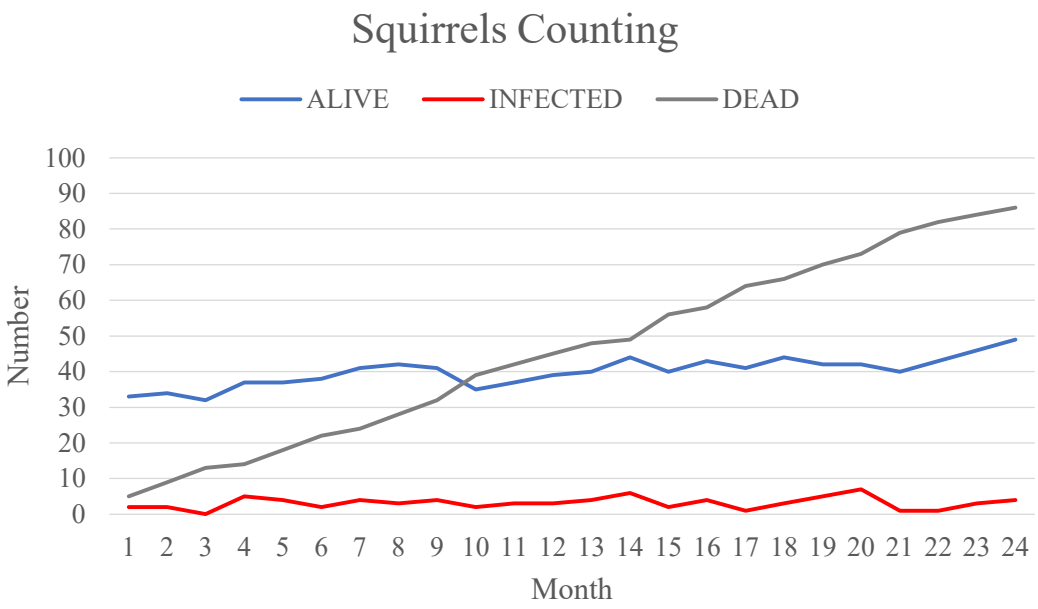

Figure 7: The number of alive, sick and dead squirrels changing through the months updating.

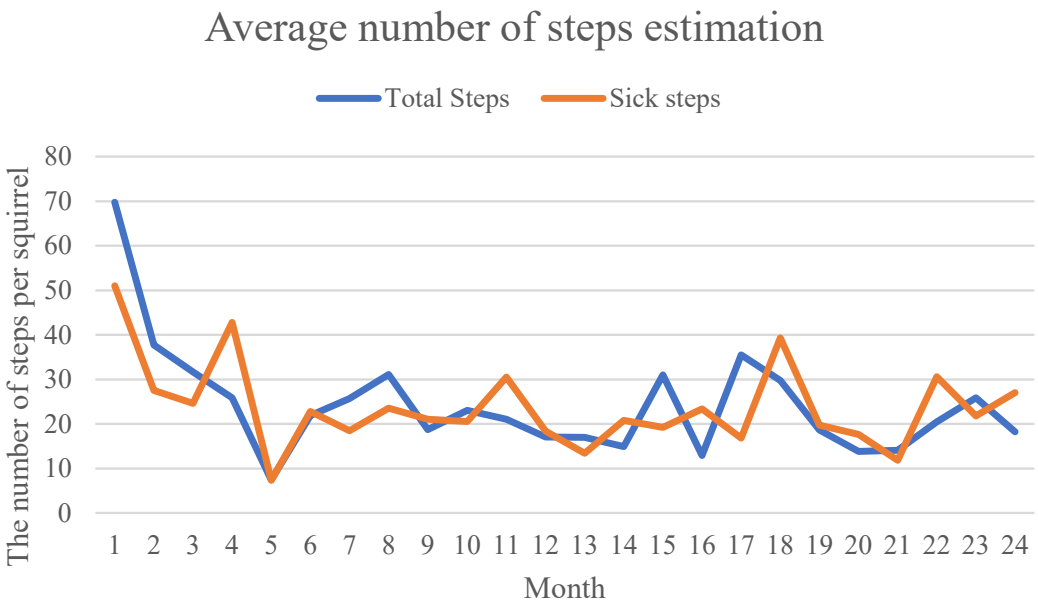

Figure 8: The curve of every number of steps through month updating.