Parallel Design Pattern Coursework2

Exam number: B129230

1. Design of Implementation

The program uses the actor pattern and 'master/worker' strategy to implement Biologist's model. There are four abstract actors in the design: *'time', 'counter', 'squirrel', 'land'*. There is only one 'master' with MPI rank number 0 and lots of 'workers' with other rank numbers. The design is based on one actor per processes.

1.1 Initialization

The master is responsible for initializing the specified number of *squirrel actors, lands actors,* a *counter actor and a time actor* and then assigning them to workers. The method of assigning jobs is that the master calls *startWorkerProcess ()* to create one new worker and call *MPI_Send ()* to assign them to the specific MPI processes. Especially, the return value of *startWorkerProcess ()* equals the MPI process rank number.

The workers representing four different types of actors to simulate Biologist' model. Moreover, it distinguishes child squirrels according to parentId, because all child squirrels are created by *the actor counter* which is rank number 18.

All actors are specified to the exact type in the *workerCode ()*. The *workerCode ()* receives initial messages with different types from the master and then receives child squirrels from *the actor counter*. There is a *switch* statement to assign them to do different jobs.

```
switch (type)
{
case _HEALTHY_SQUIRREL_TYPE:
        Actor_Squirrel ();
        break;
    case _INFECTED_SQUIRREL_TYPE:
        Actor_Squirrel ();
        break;
```

```
    case _LAND_TYPE:
        Actor_Land ();
        break;
    case _TIME_TYPE:
        Actor_Time ();
        break;
    case _COUNTER_TYPE:
        Actor_Counter ();
        break;
    default:
        printf (ERROR!!);
    break;
}
```

1.2 Actors

The Time Actor is created by the master. It is a timer to simulate 24 months and inform *16 land actors* and *the counter actor* per month. The interval between each month is to call *sleep ()* to hang up the process which call this function for microseconds.

Each month *the time actor* informs *the counter actor* to output the current active squirrels and infected squirrels, and also informs *all lands actors* to update the *populationInflux* and *infectionLevel.*

The time actor will inform *the counter actor* to stop creating new squirrels first and then *the counter actor* call *shutdownPoll ()* when the time is over. The more details about termination and time interval will explain in section 3.

```
for (i = 0; i < _MONTH; i++)
{
    sleep (1);
    if (shouldWorkerStop ()) { break; }
    MPI_Send to notify each land and the counter actor one month has passed
}
Time is over informing the counter actor to shutdownPool();
```

The Counter Actor is created by the master. It is responsible for counting the number of active, dead, infected squirrels and representing all squirrels to create child squirrels. There are two variables called *activeSquirrels* and *infectedSquirrel* in this actor for counting active and infected squirrels. For example, *the counter actor* receives the notification of giving birth and being infected from *squirrel actors*, these two variables will both plus 1. Conversely, if they receive the notification of dying, they both will reduce 1.

```
if (MPI_TAG == _NEW_SQUIRREL)
{
    activeSquirrels++;
    MPI_Recv notification and parents' position from squirrels to give birth;
    if (activeSquirrels >= _MAXIMUM_SQUIRRELS_NUMBER) {shutdownPoll ();}
    else {startWorkerProcess (); MPI_Send type and birthPlace to workerCode ()}
}

if (MPI_TAG == _DIE_SQUIRREL)
{
    activeSquirrels--;
    infectedSquirrels--;
    MPI_Recv notification from squirrels that squirrels die;
```

```
    if (activeSquirrels <= 0) {shutdownPoll ();}
}

if (MPI_TAG == _INFECTED_SQUIRREL)
{
    infectedSquirrels++;
    MPI_Recv notification and parents' position from squirrels to give birth;
}
```

There is one termination situation that *the time actor* informs *the counter actor* to terminate the program when the time is over. However, *the counter actor* also has another situation to terminate the program, which is when the number of squirrels is reduced to zero or over the maximal quantity so that t*he counter Actor* terminates the program by calling *shutdownPoll () directly*.

The processes of termination are that calling *shutdownPoll ()* to instruct the master to shut down the process pool, and each worker will either get the terminate signal when it is in workerSleep status or next calls *shouldWorkerStop ()*.

```
if (MPI_TAG == _SHUT_DOWN)
{
    MPI_Recv shutdown notification from the time actor;
    shutdownPoll ();
}
```

In addition, *the counter actor* is also responsible to receive one notification per month from the master to output current active squirrels.

```
if (MPI_TAG == _UPDATE_MONTH)
{
    MPI_Recv notification from the time actor;
    output the number of active, infected Squirrels and how many months have passed;
}
```

The land actors are 16 lands in total and created by the master. The lands' MPI process rank number is fixed from 1 to 16. The main responsibility of each land is to count and update two attributes: *populationInflux* and *infectionLevel*, and then send these two attributes back to each squirrel.

The design of counting *populationInflux* and *infectionLevel* has the almost same implementation but only one difference is the former to count all squirrels and the latter to count infected squirrels. Take counting *populationInflux* as an example, if one squirrel moves to one land, that land has a static *counterPI* variable to plus 1 every time. This variable is static for each land to count the total number of squirrels which have stepped into this land in total. There is also another static *counterIL* variable to count the number of infected squirrels and plus 1 every time. Especially, each time this land receives movement notification and it will send back *populationInflux* and *InfectionLevel* value to that squirrel who has stepped in.

```
if (MPI_TAG == _MOVE_NOTIFICATION)
{
    MPI_Recv squirrel A's information(health status) and count them
    counterPI++;
    if (squirrel.Status == _INFECTED) { counterIL++; }
    MPI_Send populationInflux and InfectionLevel back to Squirel A;
}
```

The lands also receive notification monthly from *the time actor* to update the *populationInflux* and *infectionLevel* since the 3rd months and the 2nd months respectively. Then each land will update these two attributes per month. There should be no value in the first month.

```
if (MPI_TAG == _UPDATE_MONTH)
{
    MPI_Recv time from the time actor per month);
    if (month >= 2) {populationInflux = PI[month]-PI[month-2];}
    if (month >= 1) {infectionLevel = IL[month]-IL[month-1];}
}
```

The squirrel actors are 34 squirrels at the beginning. There are four mainly things that squirrels will do: keeping moving, go dying, catching disease and giving birth.

The first thing is that squirrels will keep moving until they receive the termination. The way is continuously keeping moving and also checking the *shouldWorkerStop ()*. Each step they move, each message they will send to the specific land. For example, squirrels get the new land position by calling *getCellFromPosition ()*, they will send messages to that land. The message is an integer to represent health status of this squirrel. The destination of that land is *getCellFromPossition ()* returned value plus 1, because *getCellFromPosition ()* always returns one integer to represent land positions from 0 to 15, and lands' MPI processes rank number are constant number from 1 to 16.

After the corresponding lands receive squirrels' messages, lands will send back *populationInflux* and *infectionLevel* so that squirrels have two buffer arrays with 50 elements respectively to store the lands' messages in the past 50 steps. The messages are *populationInflux* and *infectionLevel* from *any land*.

Catching disease and giving birth are related to *average populationInflux* and *infectionLevel.* The design of calculating these two variables is almost the same. Take *average populationInflux* as an example, each squirrel will have a 50 elements buffer array to store land *populationInflux* of the past 50 steps. The average populationInflux is a summation of this buffer dived 50, because this 50 elements array can be reused from the head of the array when the buffer is full, which means this array will always store *populationInflux* of the past 50 lands. Thus, the average populationInflux always equals the summation of *populationInflux* in the entire array dividing 50.

```
if (healthyStatus == _INFECTED)
{
    MPI_recv populationInflux and infectionLevel from correspoind lands
    counterAvgPI=0;
    PopulationInfluxBuf[counterAvgPI] = populationInflux;
    counterAvgPI++; if (counterAvgPI == 50) {counterAvgPI = 0;}
}
```

Then squirrels will calculate *average populationInflux* by using this array and notify the counter actor to create child squirrels:

```
sumAvgPopulationInflux = 0;
for (i = 0; i < 50; i++)
{
   sumAvgPopulationInflux += PopulationInfluxBuf[i];
}
giveBirth = willGiveBirth(sumAvgPopulationInflux / 50, seed);
to determine whether let the counter actor to give birth;
```

Especially, it is noteworthy that only healthy squirrels can catch diseases and infected squirrels can die. Death or infection of squirrels, they will be both notified to *the counter actor.*

Each squirrel has an *infectedStep* variable to count the number of moving steps only when it is infected. If *infectedStep* variable is over 50, it will die and notify *the counter actor* and then break to call *workSleep ().*

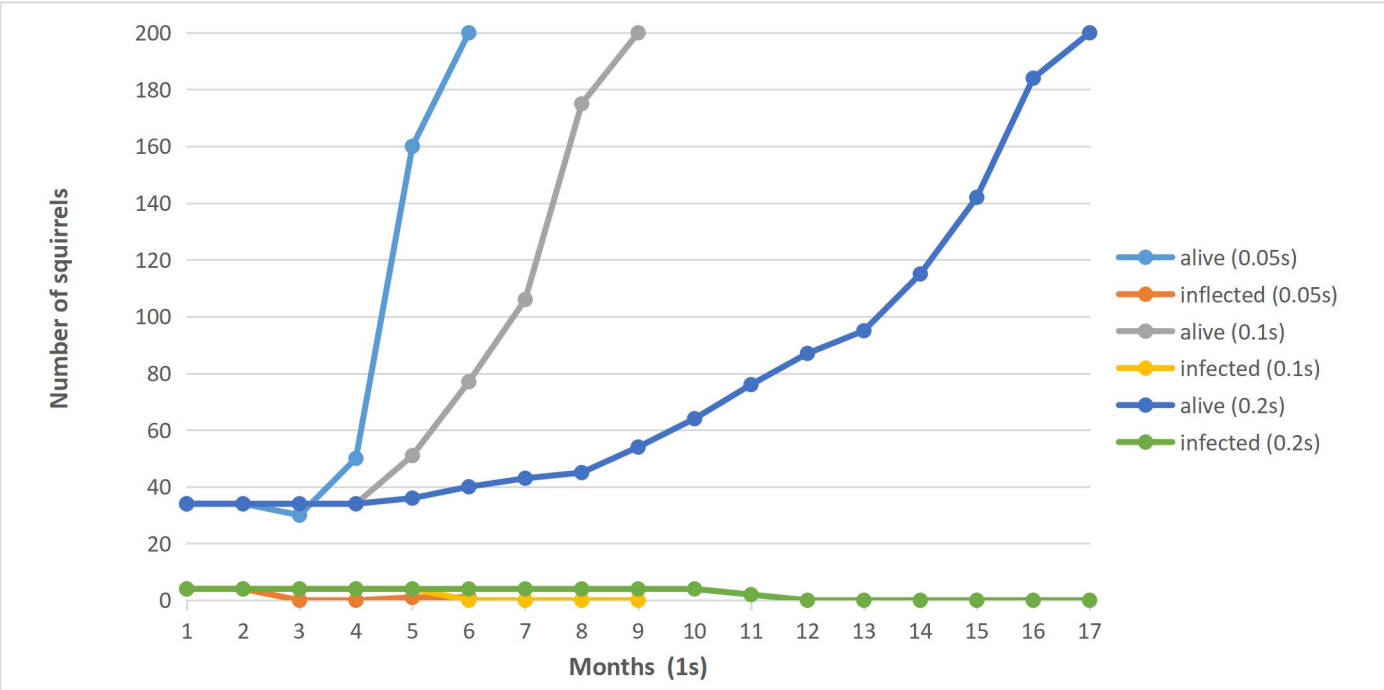
```
if (healthyStatus == _INFECTED)
{
    infectedSteps++;
    if (infectedSteps > 50)
    {
        die = willDie(seed);
        if (die == 1)
        {
            MPI_Send notification to the counter actor.
            break;
        }
    }
}
```

2. Results Analysis

2.1 Results and Discussion of Output

The program can meet all Biologist's model requirements to simulate 200 squirrels at least for 24 months, and running on 220 MPI processes without blocks and problems. The output of the simulation is population influx and the infection level for each land cell per moth, and the total number of currently alive squirrels and the number that are infected per month.

According to the analysis of biologist' simulation with 30 healthy squirrels, 4 infected squirrels and original biologist' functions, the author finds that squirrels have huge possibilities to give birth but not easy to catch diseases, which means finally the number of squirrels will over the 200 maximal quantity. In addition, the author also finds that the squirrels will unlimited reproduce finally without dying if there is no restriction of maximal quantity, because all infected squirrels all die but healthy squirrels have not caught diseases.



Graph 1: Results of simulation with 30 healthy and 4 infected squirrels

The graph shows that healthy squirrels are not easy to catch diseases and always cause unlimited reproduce. The author speculates that essential problem should be that *infectionLevel* is too low and *populationInflux* is too high, but actually it should be. If wants to break the situation of unlimited reproduce, infected squirrels should infect more healthy squirrels as far as possible within the first 100 steps. The possible methods are increasing the probability of catching diseases, decreasing probability of giving birth or adding the initial number of infected squirrels. By adjusting the parameters, this program can be adjusted to get balance within 24 months.

2.1 Flexibility and Correctness

The author's program can support more than 200 squirrels or even more to simulate Biologist's model, but the current version can not support more lands. The reason is that the counter actor, the time actor and all lands are all fixed from rank number 1 to 18. All squirrels MPI process number starts from 19 so that squirrels can be increased to the *MPI_Comm_size()* which has been set before running. The program can guarantee that all MPI processes can be terminated cleanly without blocks. The more details about termination will be explained in section 3.
The author tests the program each part step by step to check them work while coding. In addition, the author also tests the entire program after all parts finish. Thus, the author set one flag to output some important information in terms of giving birth, catching diseases and go dying by setting SL_DEBUG to 1.

The program can be divided into several important parts: squirrels movements, calculating average *populationInflux and infectionLevel,* giving birth, catching diseases and go dying.

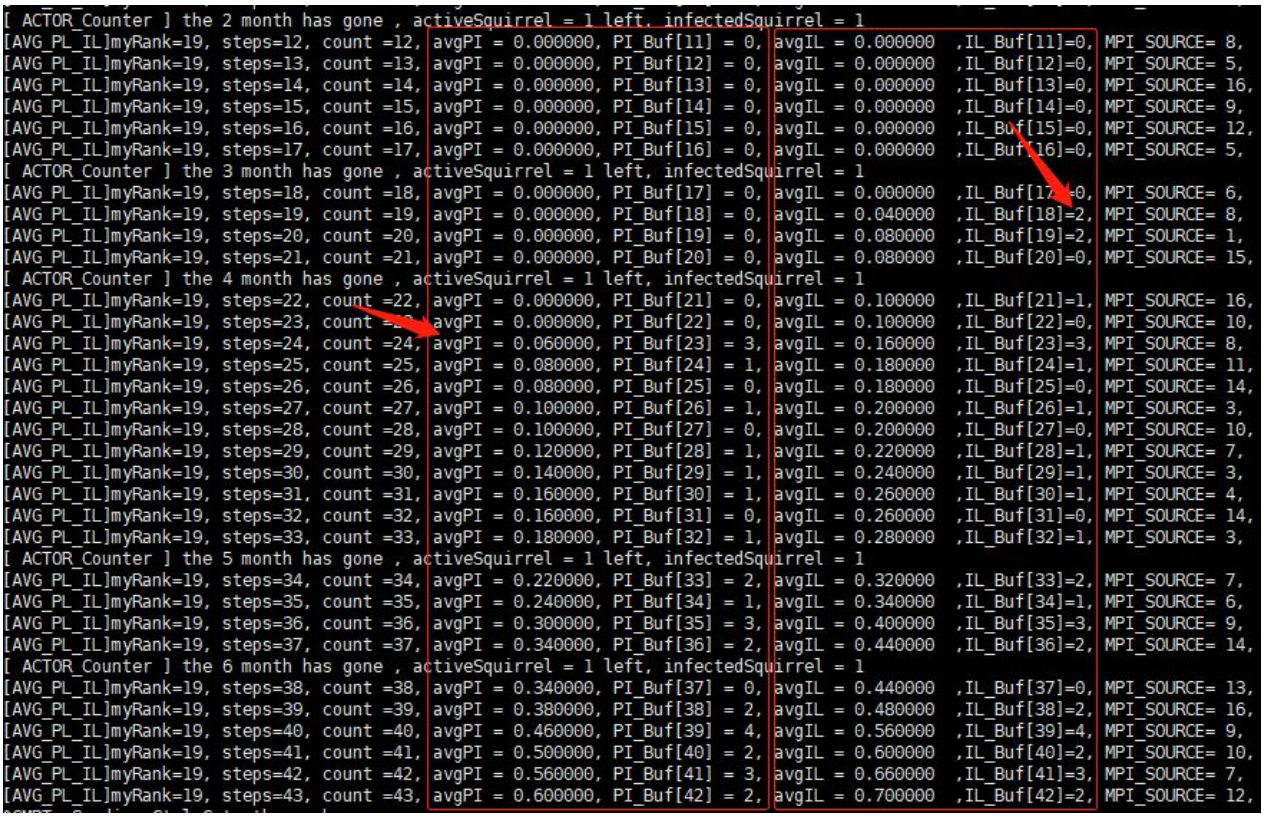
The first test is to guarantee squirrels' movement as designed and expected. The code should set `#define TRACK_STEPS_DEBUG_S2L 1, #define SL_DEBUG_L2S 1, #define PI_IL_OUTPUT 0` and only one active infected squirrel for the purpose of output clearly.

```
[LAND_RECV] myrank = 1, from Squirrel 19, Infected = 1, counterPI = 1, counterIL = 1, MPI_TAG=99
[LAND_SEND_BACK_PI & IL] myrank = 1,  populationInflux = 0, infectionLevel=0, counterPI=1, counterIL =1, MPI_SOURCE= 19,MPI_TAG=299
[SQUIRREL_MOVING] myrank = 19, steps=1, move to land =1
[SQUIRREL_RECV_PL_IL] myrank = 19, steps=1, populationInflux = 0, infectionLevel = 0, from land= 1, MPI_TAG=299

[SQUIRREL_MOVING] myrank = 19, steps=2, move to land =4
[LAND_RECV] myrank = 4, from Squirrel 19, Infected = 1, counterPI = 1, counterIL = 1, MPI_TAG=99
[LAND_SEND_BACK_PI & IL] myrank = 4,  populationInflux = 0, infectionLevel=0, counterPI=1, counterIL =1, MPI_SOURCE= 19,MPI_TAG=299
[SQUIRREL_RECV_PL_IL] myrank = 19, steps=2, populationInflux = 0, infectionLevel = 0, from land= 4, MPI_TAG=299

[LAND_RECV] myrank = 8, from Squirrel 19, Infected = 1, counterPI = 1, counterIL = 1, MPI_TAG=99
[LAND_SEND_BACK_PI & IL] myrank = 8,  populationInflux = 0, infectionLevel=0, counterPI=1, counterIL =1, MPI_SOURCE= 19,MPI_TAG=299
[SQUIRREL_MOVING] myrank = 19, steps=3, move to land =8
[SQUIRREL_RECV_PL_IL] myrank = 19, steps=3, populationInflux = 0, infectionLevel = 0, from land= 8, MPI_TAG=299

[LAND_RECV] myrank = 1, from Squirrel 19, Infected = 1, counterPI = 2, counterIL = 2, MPI_TAG=99
[LAND_SEND_BACK_PI & IL] myrank = 1,  populationInflux = 0, infectionLevel=0, counterPI=2, counterIL =2, MPI_SOURCE= 19,MPI_TAG=299
[SQUIRREL_MOVING] myrank = 19, steps=4, move to land =1
[SQUIRREL_RECV_PL_IL] myrank = 19, steps=4, populationInflux = 0, infectionLevel = 0, from land= 1, MPI_TAG=299

[ ACTOR_Counter ] the 1 month has gone , activeSquirrel = 1 left, infectedSquirrel = 1
[LAND_RECV] myrank = 11, from Squirrel 19, Infected = 1, counterPI = 1, counterIL = 1, MPI_TAG=99
[LAND_SEND_BACK_PI & IL] myrank = 11,  populationInflux = 0, infectionLevel=0, counterPI=1, counterIL =1, MPI_SOURCE= 19,MPI_TAG=299
[SQUIRREL_MOVING] myrank = 19, steps=5, move to land =11
[SQUIRREL_RECV_PL_IL] myrank = 19, steps=5, populationInflux = 0, infectionLevel = 0, from land= 11, MPI_TAG=299
```

Graph 2: Process of movements

From the picture, squirrel 19 moves to land 4, then land 4 will do *counterPI++* and *counterIL++*, then send *populationInflux* and *infectionLevel* back to squirrel 19 and finally squirrel 19 receives land 4 *populationInflux* and *infectionLevel*. This pictures shows the process of communication between squirrels and lands.

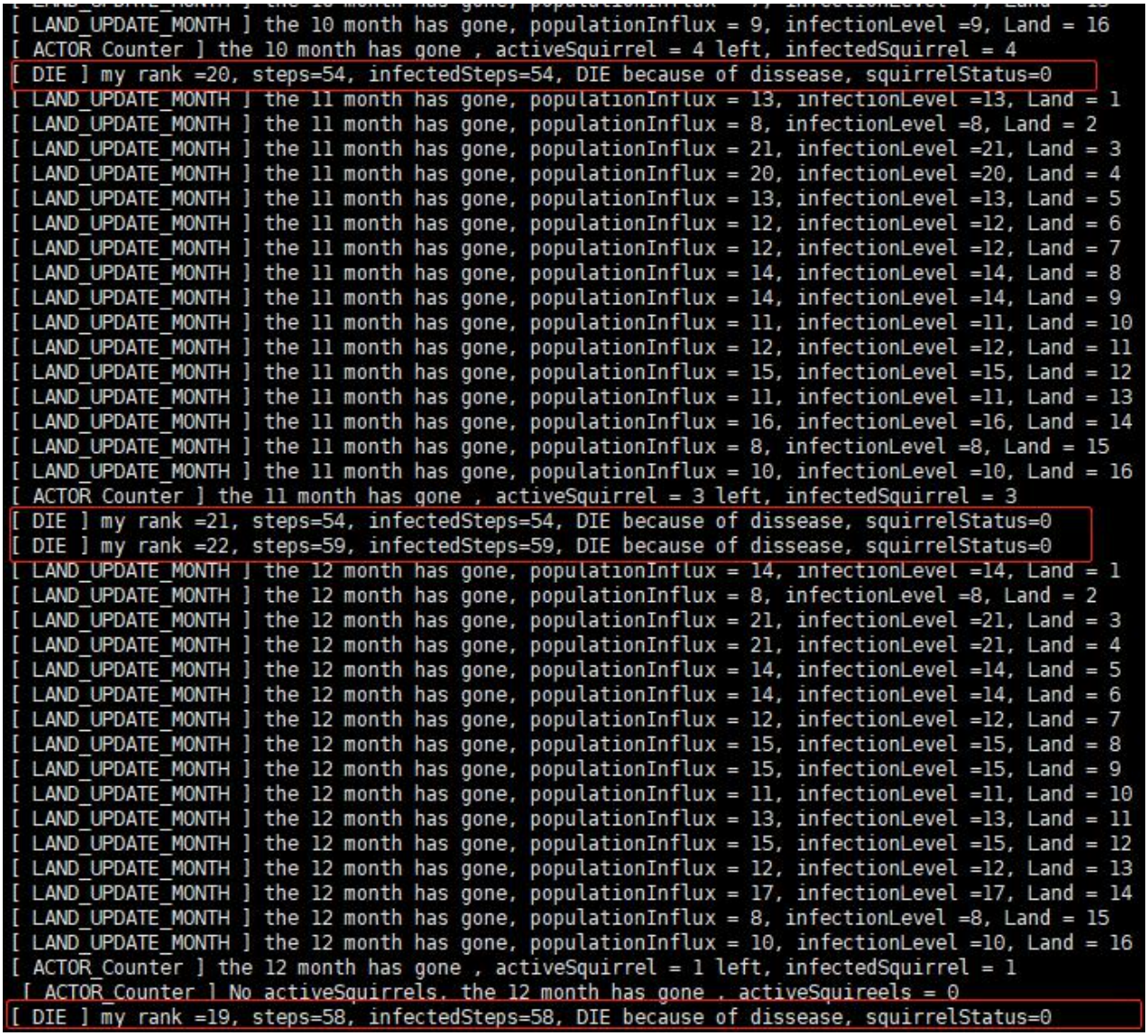
About the correctness of average *populationInflux* and *infectionLevel*, the author prints one squirrel all steps to count manually to check its correctness.



Graph 3: Average PopulationInflux and InfectionLevel

From the picture, after three months, the squirrel 19 first receives the *populationInflux* 3 from the land 8 and then calculate the *average populationInflux* 3/50 = 0.06. Then the second times receives *populationInflux* 1 from the land 11 and then calculate the *average populationInflux* (3+1)/50 = 0.08. Its design and calculating method have been described in section1.2, the *average populationInflux* always equals the summation of *populationInflux* in the entire array dividing 50. Thus, the rest can be checked in the same manner as well as *average infectionLevel.*

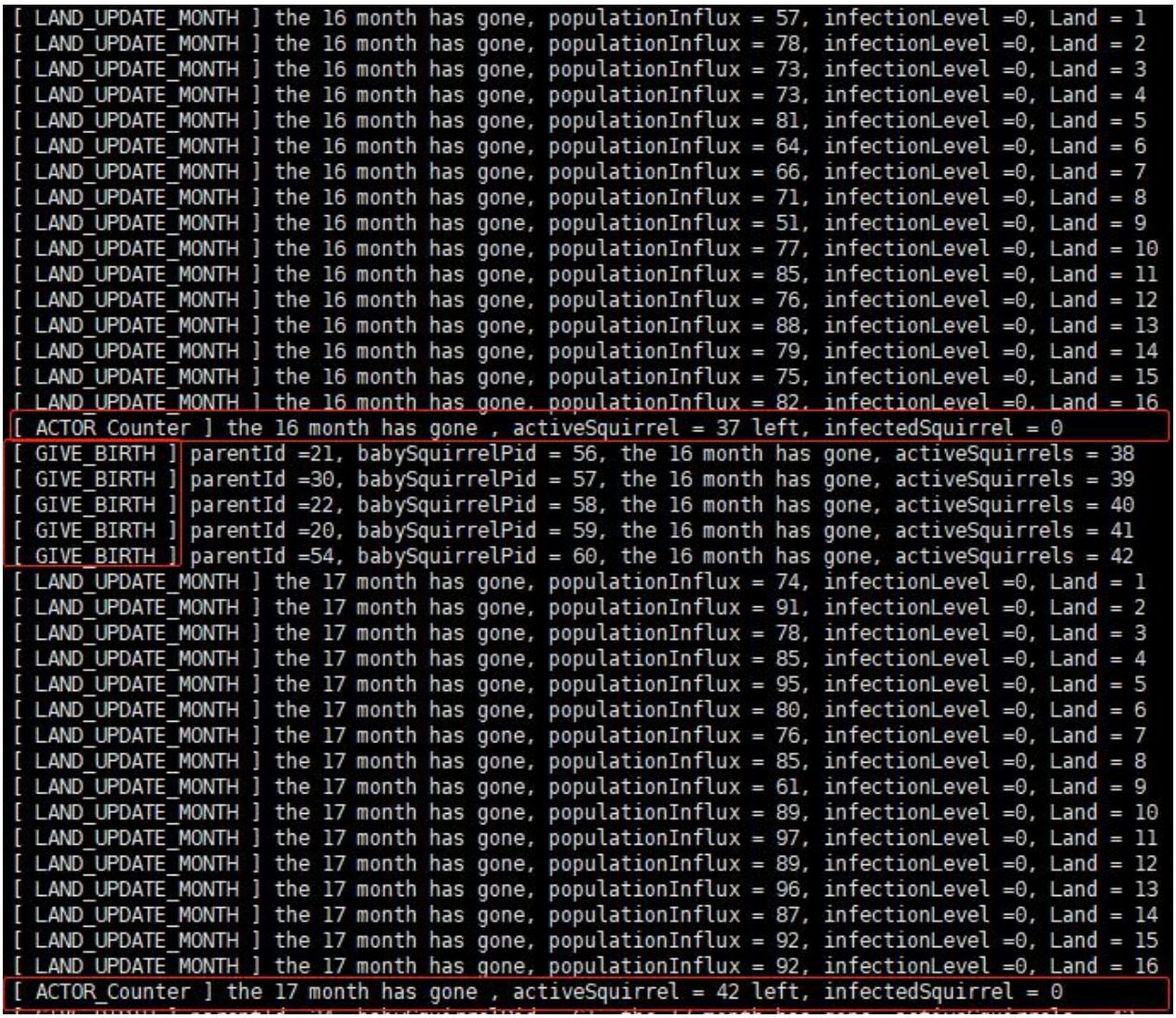
The author tests the part of *willDie ()* by tracking 4 infected squirrels as shown below.



Graph 3: Average PopulationInflux and InfectionLevel

From the picture, it proves that four infected squirrels can die one by one within 50 infected steps. In addition, the 10th month shows 4 alive infected squirrel, the 11th month shows 3 alive and the 12th month shows no squirrels alive. Then the program is terminated.

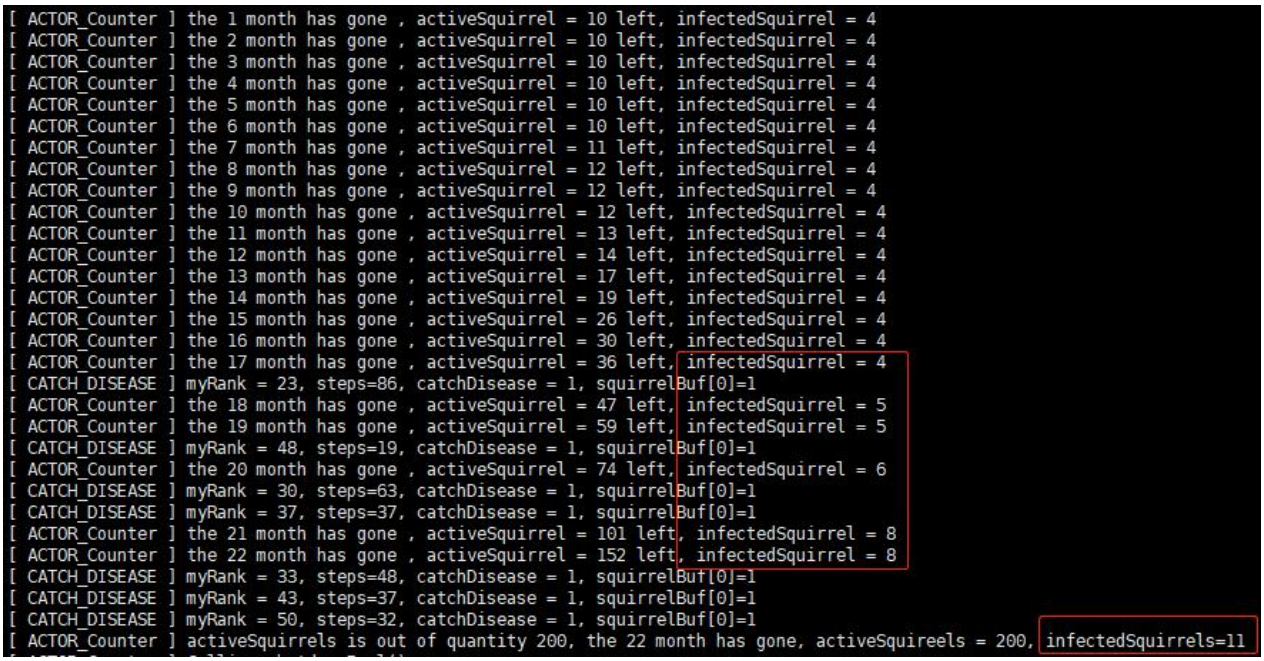
Another test only tracks the 10 healthy squirrels giving birth situation, which is the part of *willGiveBirth ()*.



Graph 4: Test of Giving Birth

As shown in the picture, the number of current alive squirrels are 37 during the 16th month. Creating 5 squirrels in the 17th month and the number of active squirrels increase to 42. In addition, this test also proves that *infectionLevel* will not be increased and healthy squirrels will not catch diseases if there is no infected squirrels.

The author tracks 4 infected squirrels and 6 healthy squirrels to check part of catching diseases, because only healthy squirrels will catch diseases and only infected squirrels can make contributions to *infectionLevel*. In this test, the author closes the part of dying to check catching diseases for the clear output.



Graph 3: Test of Catching Diseases

As the picture shown, in the 17 month, one squirrel is infected and then the 18 month, the number is increased from 4 to 5. In the 20 month, two squirrels is infected and then the 22 month, the number is increased from 6 to 8.

3. Findings and Performance Analysis

This sections mainly focus on some solutions for termination and discussion of performance.

The author spent lots of time on optimizing termination so that the submission version can guarantee all MPI processes can be finalized correctly.

About the termination, the author has tried another implementation design. There are very explicit two termination situations in this design: the first one is after 24 months later, *the time actor* will call *shutdownPoll ()* to instruct the master to shut down the process pool; the second one is when the number of squirrels is too much or too few, *the counter actor* will call *shutdownPoll ()*.

The reason why the author doesn't choose this design because the author finds that when time interval is short for each month, these two termination situations design will probably cause program block. The phenomenon is that if the program calls *processPoolFinalise ()* before *the counter actor* closing, the program will be blocked, which means some MPI processes cannot reach *MPI_Barrier ()* of *processPoolFinalise ()*. The author speculates the *counter actor* still creates new squirrels after *the timer actor* calling *shutdownPool ()*. According to debugging, the main reason is because time interval of each month is too short so that some MPI processes can not response in time.

The author wants to avoid this problem so only keeps one termination place of the *counter actor* in the whole program to guarantee to close *the counter actor* successfully before calling *ProcessPoolFinalise()*. This design can guarantee even time is too short, the program will not be blocked.

In addition, another best solution is increasing the time interval and reduce squirrels' movement frequency. The author finds that if the time interval is too short, some workers totally no *printf ()* at all but it should have. The author speculates that the time interval can not set too short because some workers haven't been started and it also can not set too large otherwise the number of squirrels will over the maximal squirrel quantity at the first few months. Thus, the author increases the time interval, but the following problem is that squirrels moves too many steps during the whole simulation, because the author continuously calls *squirrelStep ()* and *getCellFromPosition ()* in the *while* loops without any restrictions. Each movement sending one message and high CPU frequency cause the program out of control and too many times send, the problem of message delay will be a serious problem here. Thus, the author decides to extend time interval and slow squirrels' movements to make them manageable. The practice has proven that This solution and design can make the program support more processes to increase the scalability.

On the other hand, performance is also an important consideration point. The current design of sending average populationInflux and infectionLevel has been describe in section 1.2. It is based on squirrels steps. Another optimization design idea is that all lands will send updated populationInflux and infectionLevel to all squirrels per month, so lands should have all MPI processes rank number of all squirrels. And each squirrel has an array to store all lands' populationInflux and infectionLevel. When they move to new lands, they check and count the populationInflux and infectionLevel in that local array rather than receiving these two variables from the lands which they have stepped in. If squirrels' movement frequency is very high, the communication is huge. Thus, this could be an optimization point in the future.

4. Conclusion

The final program can meet all Biologist's model requirements to simulate 200 squirrels at least for 24 months, and running on 220 MPI processes without blocks and problems, although the result of simulation is controversial.

The output of the simulation consists of population influx and infection level for each land cell per month, and the total number of currently alive and infected squirrels per month.

According to the analysis of biologist' simulation with 30 healthy squirrels, 4 infected squirrels and original biologist' functions, the author finds that squirrels have more possibilities to give birth but not easy to catch diseases. In other words, this biologist's model with 4 infected and 30 healthy squirrels will evolve into unlimited reproduce without dying finally.