Parallel Design Patterns II

B119172

April 3, 2018

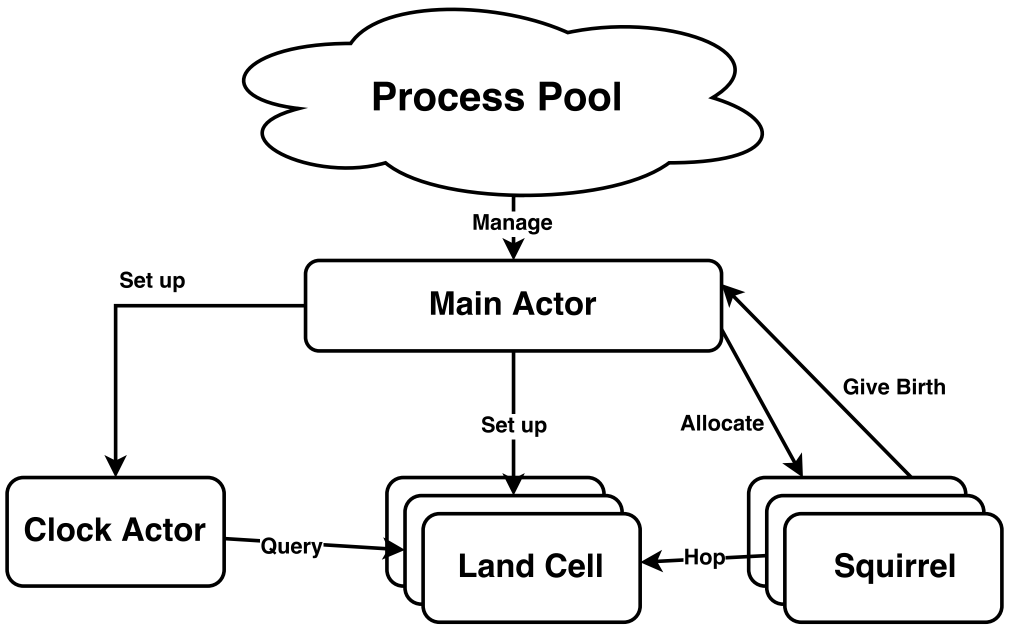
1 Description

The purpose of this document is to demonstrate the design and implementation about the biologist's model. The report will then introduce the actor framework implemented and used in the project.

2 Solution

2.1 Design

Actor pattern is used to parallelize the solution. In this case, we maintain three major actors: Clock actor, Land-cell actor and Squirrel actor. Besides, a main actor is used to manage the allocation of other actors. The structure of this program is presented below:



2.2 Implementation

Project Structure

The repository of this project can be found on https://github.com/Yiiinsh/MPI-Actor.

In this project, there are two major directories: **actor_framework** and **demo_squirrel_solution**. The (actor_framework) directory contains the implementation of actor framework written with MPI. Generally, it use a master process to maintain the process pool to do the allocations of other actors and leave all the specified work to be done on customized actors. The **demo_squirrel_solution** directory is composed of the squirrel problem specified codes. Within the directory we provide the definition of all our actors.

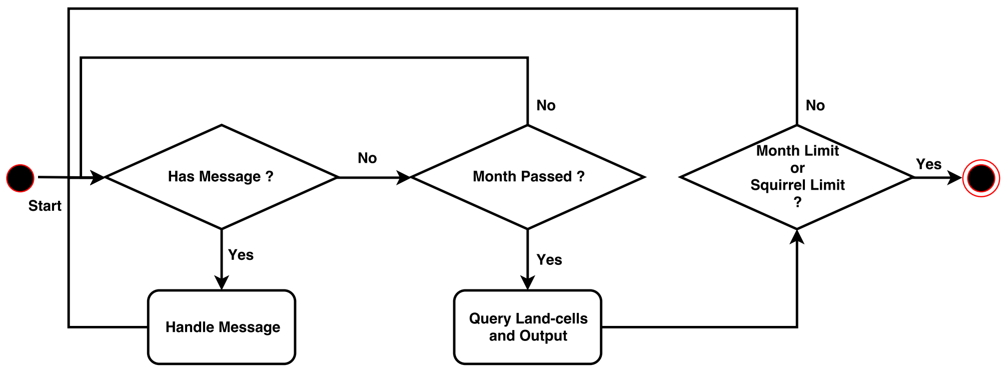
The program is compiled using gcc version 4.8.5 with GNU99 as the standard. A makefile is provided to build the project.

Main Actor

Main actor is responsible for the management of process pool. In the project, according to the configurations in **solution_configuration.h**, main actor will initialize the clock actor, land cell actors and squirrel actors respectively. After the initialization phase, main actor will call the **masterPoll** function and wait for further instructions to allocate other actors.

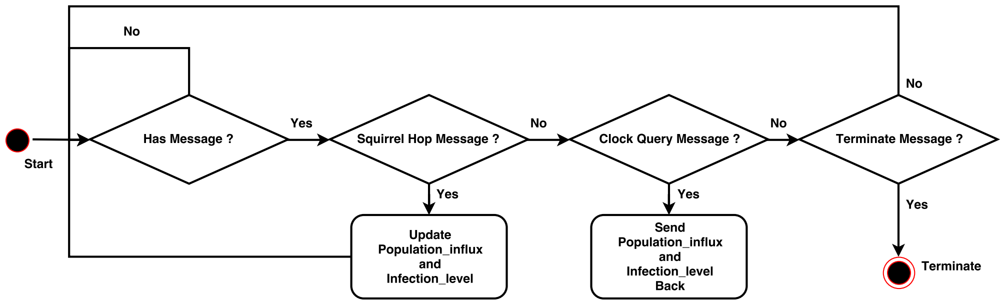
Clock Actor

The clock actor is account for the management of global time. As time goes on, the clock actor will query land cell actors for current statistics. Besides, the clock actor will receive the message from squirrel actors to calculate the number of living squirrels and infected squirrels. The execution diagram of Clock Actor can be seen in the following figure:



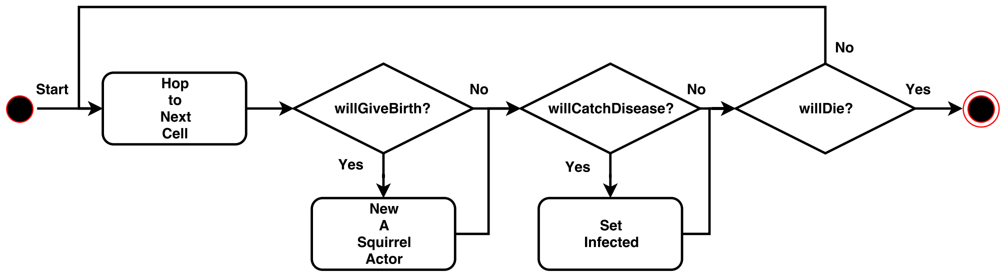
Land-cell Actor

The land cell actor represents a single unit of land cell which can host squirrels. It can receive hop message from squirrels and update its population_influx and infection_level. Besides, each land cells will receive a query request from clock actor every month and it will send current corresponding statistics back to the clock actor. The execution pattern of land cell actor is shown in the following figure:



Squirrel Actor

The squirrel actor stands for a squirrel entity. It can move across land cells until it dies. Besides, it can give birth to other squirrels by sending a command to main actor and asking for allocation of another squirrel actor. The execution flow of squirrel actor is demonstrated in the following figure:



2.3 Results

The demonstration application has been run with the suggested configuration, the results of first three month are presented below and details of the result can be found on the project repository:

```
[MONTH    1]  Health: 30, Infected: 4, Total: 34
landcell              population_influx       infection_level
1                     19                      3
2                     18                      1
3                     17                      1
4                     12                      1
5                     21                      0
6                     26                      10
7                     26                      8
8                     21                      2
9                     18                      5
10                    26                      4
11                    26                      1
12                    19                      4
13                    23                      3
14                    25                      5
15                    24                      9
16                    22                      4
```

```
[MONTH    2] Health: 31, Infected: 2, Total: 33
landcell                 population_influx        infection_level
1                        60                       7
2                        72                       8
3                        58                       5
4                        63                       5
5                        67                       2
6                        64                       12
7                        73                       12
8                        75                       10
9                        55                       9
10                       89                       14
11                       69                       5
12                       78                       16
13                       75                       8
14                       73                       10
15                       75                       17
16                       70                       9

[MONTH    3] Health: 30, Infected: 1, Total: 31
landcell                 population_influx        infection_level
1                        108                      13
2                        111                      10
3                        113                      7
4                        106                      7
5                        119                      9
6                        109                      9
7                        111                      8
8                        115                      10
9                        101                      6
10                       137                      21
11                       114                      9
12                       122                      17
13                       129                      10
14                       119                      7
15                       118                      10
16                       121                      11
```

3 Actor Framework

3.1 Introduction

This framework is implemented in ”one actor per process” pattern. All processes are maintained in a process pool and users can apply this framework to solve specified problem by implementing their own actors.

3.2 Design

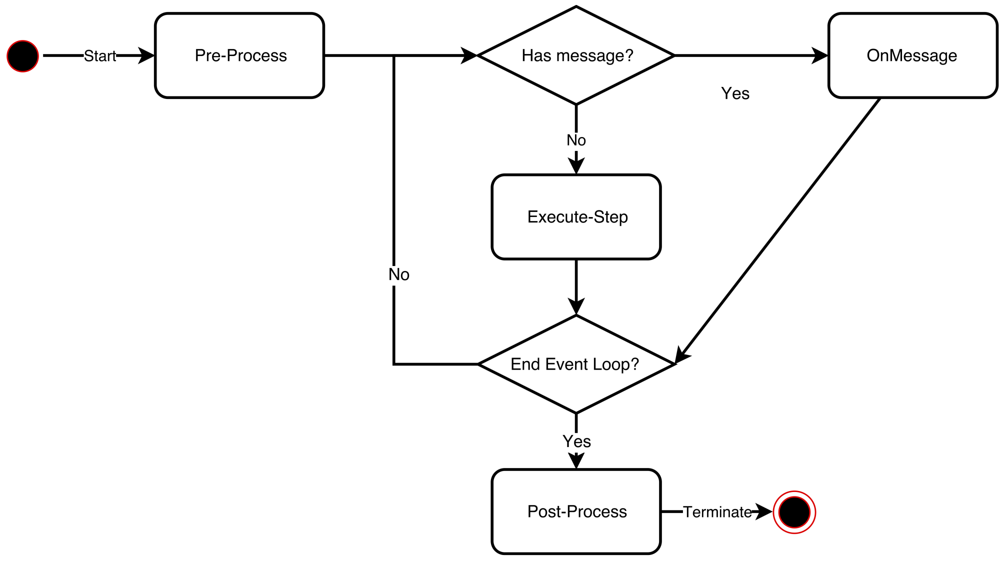
The definition of actor used in the framework can be abstracted as:

```
typedef struct __actor {
    char type[ACTOR_TYPE_NAME_LIMIT];
    bool event_loop;

    void (*on_message)(struct __actor *self, MPI_Status *status);
    void (*execute_step)(struct __actor *self, int argc, char **argv);
    void (*new_actor)(struct __actor *self, char *type, int count);
    void (*pre_process)(struct __actor *self);
    void (*post_process)(struct __actor *self);
    void (*terminate)(struct __actor *self);

    } ACTOR;
```

Users of the framework can create their own actors with the customized life cycle hooker. After creation, every actor will run through the pattern:



The entry of the program is provided in the framework within **main.c**. It runs in the following way:

```
statusCode = processPoolInit();
if (2 == statusCode) /* Master actor */
{
    create_main_actor(&actor);
    actor_start(&actor);
}
else if (1 == statusCode) /* Worker Actor */
{
    // Recv actor type
    char type[ACTOR_TYPE_NAME_LIMIT];
```

```
        memset(type, '\0', ACTOR_TYPE_NAME_LIMIT);
        MPI_Recv(type, ACTOR_TYPE_NAME_LIMIT, MPI_CHAR, RANK_MAIN_ACTOR,
                ACTOR_CREATE_TAG, MPI_COMM_WORLD, MPI_STATUS_IGNORE);

        // create actor and start
        create_actor(type, &actor);
        actor_start(&actor);
    }
```

The process on Rank 0, also known as the main actor, are treated as the master of this pool, which is responsible for the allocation of other actors. A template for main actor is provided and users can further customized it on the pre-process stage to do some set up works for the application. After completion of all the set up works, the master will only serve as the pool master and wait for further commands about the pool:

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

In the framework, **IProbe** is used to detect incoming messages, users can customized as their needs with different tags to denote different message.

3.3 Usage

The main entry of the application are provided in the framework. For a better usage of this framework, there are some best practice to follow:

- Understand the requirements and design corresponding actors
- Implement those actors in the format defined in **actor.h**, use different tags to denote messages for different purpose
- Modify the **main_actor_template.h**, customize the pre-process function to set up system environments (initialize original actors)
- Implement the interface declared in **customized_actors.h**, it is a guidance to the framework about how to create new actors
- Write a corresponding Makefile for the solution
- Build up the project and run it, it is users' responsibility to ensure that enough processes have been allocated for MPI