# Parallel Computations for Large-Scale Problems
# sf2568

Project Report

Yiming Fan
yimingf@kth.se

## 1 Problem Description

This project aims at simulating a parallelized cellular automata of the classic predator-prey model. Since migrations occur on predators and preys some communications among cells need to be implemented.

Specifically this problem comes out of the textbook of the course, *Parallel Programming* by Barry Wilkinson, Michael Allen. Below is the shortcut of the problem description:

*Once upon a time there was an island populated only by rabbits, foxes, and vegetation. Some local geographers have even drawn gridlines that serve to divide the island into squares to facilitate their demographic studies on the populations of each inhabitant.*

*Within each square the populations of rabbits and foxes are governed by several factors: 1. the population of foxes and rabbits, 2. the reproduction rates, 3. the vegetation growth rate, 4. the death rates of 'old' rabbits, 5. the eating habits of rabbits and foxes and 6. the migration of rabbits and foxes. Our job is to simulate some long time of life on the island given some initial conditions.*

We consider the problem as an advanced version of *Game of Life*. While some factors are affected by random number generators, the system follows the definition of cellular automata and can be computed cell-wise with little communication with adjacent cells. The specified algorithm is shown below.

## 2 Description of the Algorithm

As stated in the problem description at each 'day' we need to calculate by order:

1. the reproduction of predators and preys (on some 'birthing' days)
2. the consumption of vegetation
3. the consumption of prey
4. the natural death of predators
5. the natural death of prey, and
6. the `migration`.

All factors except the `migration` could be calculated cell-wise independently, hence we will have a `beforeMigration` process which could be implemented by embarassingly parallelization.

We implemented the `migration` process by the following algorithm (below pseudocode is `c/python`-like):

```
// before migration
migrationFox = malloc(sizzeof(int)*(x+4)*(y+4))
// since foxes can arbitrarily migrate 2 cells far away
// we created the matrix with 2 extra borders
migrationRabbit = malloc(sizeof(int)*(x+2)*(y+2))
// since rabbits can arbitrarily migrate 1 cell far away
// we created the matrix with 1 extra border

// begin migration
for (each cell c) in process #p:
    migratedFoxes = 0
    migratedRabbits = 0
    neighbors = adjacentCell(c, fox) // 8 neighbors
    for (each neighbor n) in neighbors:
        foo = random()*(c.foxes*0.125) // ensure c.foxes>0
        migrationFox[n] += foo
        migratedFoxes += foo
    end for
    c.foxes -= migratedFoxes

    neighbors = adjacentCell(c, rabbit) // 4 neighbors
    for (each neighbor n) in neighbors:
        foo = random()*(c.rabbits*0.05) // not exceeding 20% of rabbits
        migrationRabbits[n] += foo
        migratedRabbits += foo
    end for
    c.foxes -= migratedFoxes
end for

// begin communication
for (each neighbor process n) in process #p: // non-blocking send
    MPI_Isend(direction=sendDirection(p, n), edge, terminal=n)
end for
for (each neighbor process n) in process #p: // non-blocking receive
    MPI_recv(direction=receivingDirection(p, n), edge, terminal=n)
end for

// updage from received edge data.
for edge in edges:
    (migrationFoxes, migrationRabbits) = updateEdge(edge)
end for

for (each cell c) in process #p:
    (c.foxes, c.rabbits) = updateMigration(MigrationFoxes, migrationRabbits)
end for

MPI_barrier(MPI_COMM_WORLD)
// end migration
```

At the end of the last day we gather data (number of rabbits, foxes and vegetation) from all processes and print out the results.

## 3 Theoretical Performance Estimation

### 3.1 Speeup Analysis

Denote the number of cells by #(data) = $MN$ and the number of processors $R = PQ$. We assume a perfect load balanced distribution:

$$I_p \approx \frac{M}{P}, \; J_q \approx \frac{N}{Q} \tag{1}$$

from the lecture slides of this course we know that the total communication time is:

$$t_{comm} \approx (C(P) + C(Q))t_{startup} + 3\,t_{data}(C(P)I_p + C(Q)J_q) \tag{2}$$

where

$$C(P) = \begin{cases} 0, \text{ if } P = 1 \\ 2, \text{ if } P = 2 \\ 4, \text{ if } P \geq 3 \end{cases} \tag{3}$$

$$C(Q) \text{ similar to } C(P) \tag{4}$$

For the computation time we denote the time needed for single cell by $t_{a,beforeMig}$ the time before migration (embarrassingly parallel), $t_{a,Mig}$ the time for computing migration. Apparently we have

$$t_a = t_{a,beforeMig} + t_{a,Mig} \tag{5}$$

which is same between sequential and parallelized architectures. Hence

$$tcomp = \alpha \frac{MN}{PQ} t_a = \alpha \frac{MN}{PQ} \left(t_{a,beforeMig} + t_{a,Mig}\right) \tag{6}$$

where $\alpha$ is a constant here. The best sequential time is

$$T_s^* = \alpha MN\, ta \tag{7}$$

Similar to the conclusion in the lecture, the theoretical speedup is then:

$$\begin{aligned} S_R = S_{PQ} &= \frac{T_s^*}{T_R} && (8) \\ &\geq R\,\frac{\alpha MN t_a}{\alpha MN t_a + 8\,R t_{startup} + 12(QM + PN)t_{data}} && (9) \\ &\geq R\,\frac{1}{1 + \frac{8\,R t_{startup}}{\alpha MN t_a} + \frac{12}{\alpha}\left(\frac{P}{M} + \frac{Q}{N}\right)\frac{t_{data}}{t_a}} && (10) \end{aligned}$$

## 3.2 Memory Analysis

We simply ignore the size of excessive variables (like `flag`, `numberOfSomething`) for simplicity. The memory used in the sequential program is $M_s = 3\,MN * \texttt{sizeof(int)}$. In the parallelized program the increased allocation of memory (for storing migration data) is:

$$M_m = \texttt{sizeof(int)} * [3MN + (PQ) \cdot ((I_p + 2)(J_q + 2) + (I_p + 4)(J_q + 4))] \qquad (11)$$

The memory allocated for communication is:

$$M_c = \texttt{sizeof(int)} * [3PQ \cdot (I_p + J_q)] \qquad (12)$$

Hence the total memory needed for a parallelized program is:

$$\begin{aligned} M_p &= M_m + M_c + M_s && (13) \\ &= \texttt{sizeof(int)} * [6MN + (PQ) \cdot (2I_pJ_q + 9(I_p + J_q) + 20)] && (14) \end{aligned}$$

# 4 Implementational Details

## 4.1 The Communication Architecture

The communication process takes equal time as *2-dimensional red/black communication*, as non-blocking `MPI_Isend` functions are called. No extra time is elapsed when waiting for receiving `edge` data. We will analyze the speedup performance later.

Figure 1 shows the communication structure. The migration matrix is 2 cells (for rabbits) and 4 cells (for foxes) larger than the cell size of a certain process. The process always send the `edge` data with the opposite `tag` in `MPI_Isend` function. For example if a process sends the migration data across the *west* border, it enclose the `edge` array with tag `E_UPDATE` since the array will be updated to the east border of its adjacent process. The communication size will be `3*sizeof(int)*y1` and `3*sizeof(int)*x1` depending on the direction.

## 4.2 Some tricks to reduce memory / improve speeds

- The vegetation should be stored as `double` (8 bytes) as in the original problem description. In practice we used 2-byte `int` instead for efficient storage. The variable range from 100 to 1000 which is equivalent to 0.1 and 1.0.
- The numbers of foxes and rabbits ranges from 0 to 65535. In practice no more than 1000 rabbits could survive, since every rabbit consumes 1 unit of vegetation every day.

# 5 Results of a Typical Program Run

By `make` we will get a executable file. Then by typing

```
mpirun -n (#processors) out (input file)
```

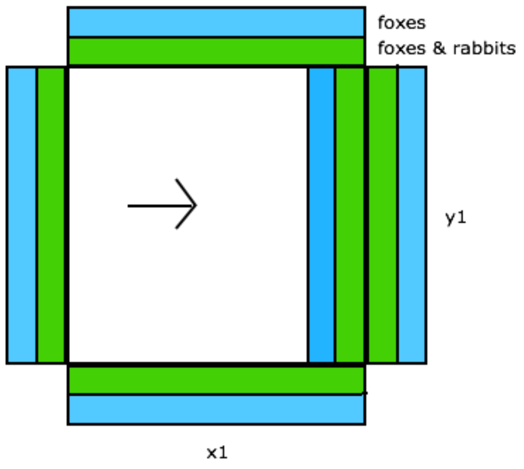

Figure 1: The communication architecture.

the result will be printed onto the terminal after the program ends. For example consider a small $8 * 8$ grid with the configuration in `TextBoard.txt`:

```
10000 // number of days
8 // size of grid
8
20 10 300 // (#foxes #rabbits #vegetation)
0 10 300
0 10 300
...
0 10 300
```

after 10000 days (that is approximately 30 years) the results are shown.

```
Yimings-MacBook-Air:p y$ mpirun -n 2 out TestBoard.txt
we have 2 partitions
0 days has gone.
1000 days has gone.
2000 days has gone.
3000 days has gone.
4000 days has gone.
5000 days has gone.
6000 days has gone.
7000 days has gone.
```

```
8000 days has gone.
9000 days has gone.

Final board config:
0 0 0 0 0 0 0 0 // grid of foxes.
0 0 0 0 0 0 0 0 // foxes usually die off very early.
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
19 20 18 17 22 17 18 20 // grid of rabbits.
19 17 16 24 16 22 20 20
18 22 18 21 20 18 21 19
19 19 19 20 16 19 18 19
22 14 30 9 13 35 40 6
19 72 69 52 36 65 94 78
81 16 22 89 94 81 7 79
56 75 0 76 62 36 99 41
1000 1000 1000 1000 1000 1000 1000 1000 // grid of vegetation.
1000 1000 1000 1000 1000 1000 1000 1000
1000 1000 1000 1000 1000 1000 1000 1000
1000 1000 1000 1000 1000 1000 1000 1000
100 100 1000 100 100 100 1000 138
1000 1000 1000 100 1000 1000 1000 1000
1000 100 1000 1000 1000 1000 368 1000
1000 1000 1000 1000 1000 1000 1000 1000
elapsed time: 83ms
```

## 6 Experimental speedup investigation with meaningful data

### 6.1 Speedup in Practice

We have tried $100 * 100 = 10000$ grid and have simulated the system for 300000 iterations (days). In general no meaningful data could be produced since the number of iterations is too large. Foxes are bound to die off and the remaining rabbits on each cell is normally 10 to 20.

We have simulated the system with number of processes ranging from 1 to 20 and a plot of the relation between the elapsed time and the number of process is also shown. The decrease in elapsed time plateaus as the number of processes increases from 10 to 20.

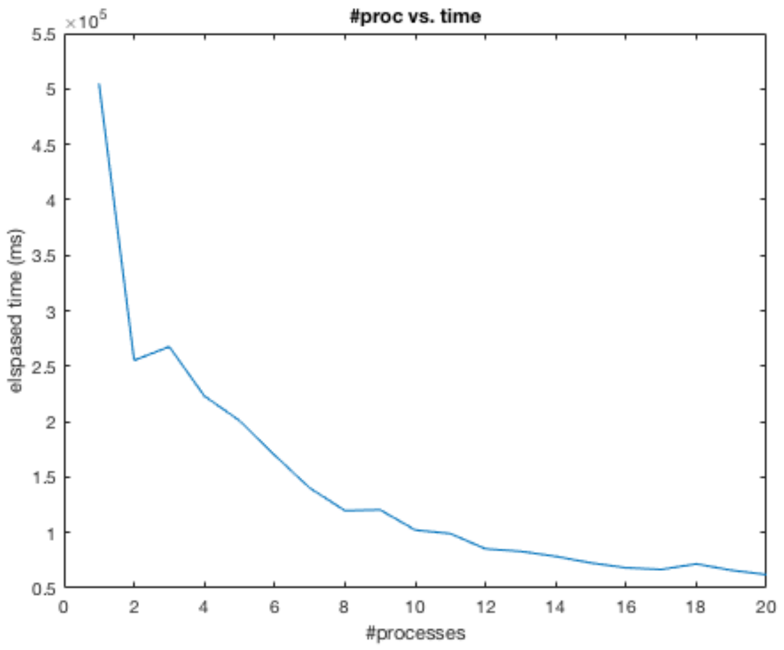

Figure 2: The number of processes vs. time.

## 6.2 Case Studies

We have tried 3 cases in the textbook:

1. Uniformly there are 2 foxes and 100 rabbits per square initially; the vegetation level is 1.0 everywhere.

2. There are 20 foxes in one corner square and none elsewhere; there are 10 rabbits in every square except in the corner square diagonally opposite the foxes, and it contains 800 rabbits; the vegetation is 0.3 everywhere.

3. There are no foxes on the island, but there are two rabbits in each square; the initial vegetation level is 0.5 everywhere.

As indicated in the textbook we tried $8 * 8 = 64$ small grids and simulated the system for 10 years (3650 days). The results are shown:

### 6.2.1 Case 1

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1000 1000 1000 1000 1000 1000 1000 1000
1000 1000 1000 1000 1000 1000 1000 1000
1000 1000 1000 1000 1000 1000 1000 1000
1000 1000 1000 1000 1000 1000 1000 1000
1000 1000 1000 1000 1000 1000 1000 1000
1000 1000 1000 1000 1000 1000 1000 1000
1000 1000 1000 1000 1000 1000 1000 1000
1000 1000 1000 1000 1000 1000 1000 1000
```

At the first few months the number of foxes increased drastically and then starved due to lack of rabbits. Rabbits extincted later.

### 6.2.2 Case 2

```
0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
18 19 20 20 17 30 21 27
19 25 22 28 31 28 29 35
29 31 31 36 31 24 32 31
33 28 31 31 36 32 29 30
35 32 32 36 25 36 26 26
32 34 26 28 41 28 30 19
30 43 26 31 24 21 23 19
28 23 22 19 21 18 18 20
1000 1000 1000 1000 1000 1000 1000 1000
1000 1000 1000 1000 1000 1000 1000 1000
1000 1000 1000 1000 1000 1000 1000 1000
1000 1000 1000 1000 1000 1000 1000 1000
1000 1000 1000 1000 1000 1000 1000 1000
```

```
1000 1000 1000 1000 1000 1000 1000 100
1000 1000 1000 1000 1000 1000 100 100
1000 1000 1000 1000 1000 1000 100 100
```

After 10 years there are only 1 fox left (and surely will die off since 1 fox cannot reproduce itself) but there are some rabbits remaining on the island.

#### 6.2.3 Case 3

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
18 20 24 22 17 36 25 30
18 28 20 31 26 21 30 27
29 28 26 32 30 32 36 33
28 38 29 40 34 34 38 36
31 31 32 34 31 30 28 29
32 33 27 31 33 29 33 30
31 32 43 31 27 28 25 19
35 35 20 17 23 16 21 19
1000 1000 1000 1000 1000 1000 1000 1000
1000 1000 1000 1000 1000 1000 1000 1000
1000 1000 1000 1000 1000 1000 1000 1000
1000 1000 1000 1000 1000 1000 1000 1000
1000 1000 1000 1000 1000 1000 1000 1000
1000 1000 1000 1000 1000 1000 1000 1000
1000 1000 1000 1000 1000 1000 1000 1000
1000 1000 1000 1000 1000 1000 1000 1000
```

This case study allows us to see how rabbits reproduce themselves after a few years, and how rabbits are served by vegetation. Since we have lowered the life span of each rabbits in practice, the real-life situation might be different from the result above.

## 7 Conclusion

We have implemented an advanced parallelized version of predator-prey model, or '*Game of Life*'. We have applied non-blocking send/receive communication architecture and the time performance reached a satisfactory level when the number of processors is above 10. We have also simulated some real-life based case studies.