MPI: Project - Game of Life

Yanwen Xu

June 2020

Abstract

In this project, I have written the 1D column decomposition(Pass level) and the 2D tile decomposition(Giga version) and Parallel I/O (Peta version).

1 PCAM

I went through the following PCAM process when designing my code:

- To start, I **Partitioned** the Game-of-Life(GoF) in domain. I decomposed the computation and data into small tasks in two ways: *1D Column Decomposition* and *2D Decomposition*. For 1D decomposition, I split the 2D GoF grids to columns. For 2D decompostion, I split the GoF grids into rectangle tiles.
- For 1D decomposition, I created asynchronous **Communication** channels for each small task to send and receive data(Ghost cells) from its left and right neighbor. For 2D decomposition, each small task will need to send/receive data from all of its 8 neighbor tiles.
- For **Agglomeration**, I evaluated my task-channel communication structures of both 1D and 2D. To reduce necessary communication, I only send the ghost cells (like the left/right most column in 1D) rather than the entire board to other processor. For more comprehensive evaluation of 1D vs. 2D, see Section 4. I used *MPI_Scatter* and *MPI_Gather* to distribute smaller task to process from the main thread.
- For **Mapping** tasks to processors: due to the limited time in development, I chose it to be completely static, and **assume** the dimension of the GoF grids is divisible by the number of processors. So that I can easily assign equal amount of smaller grids to each processor.

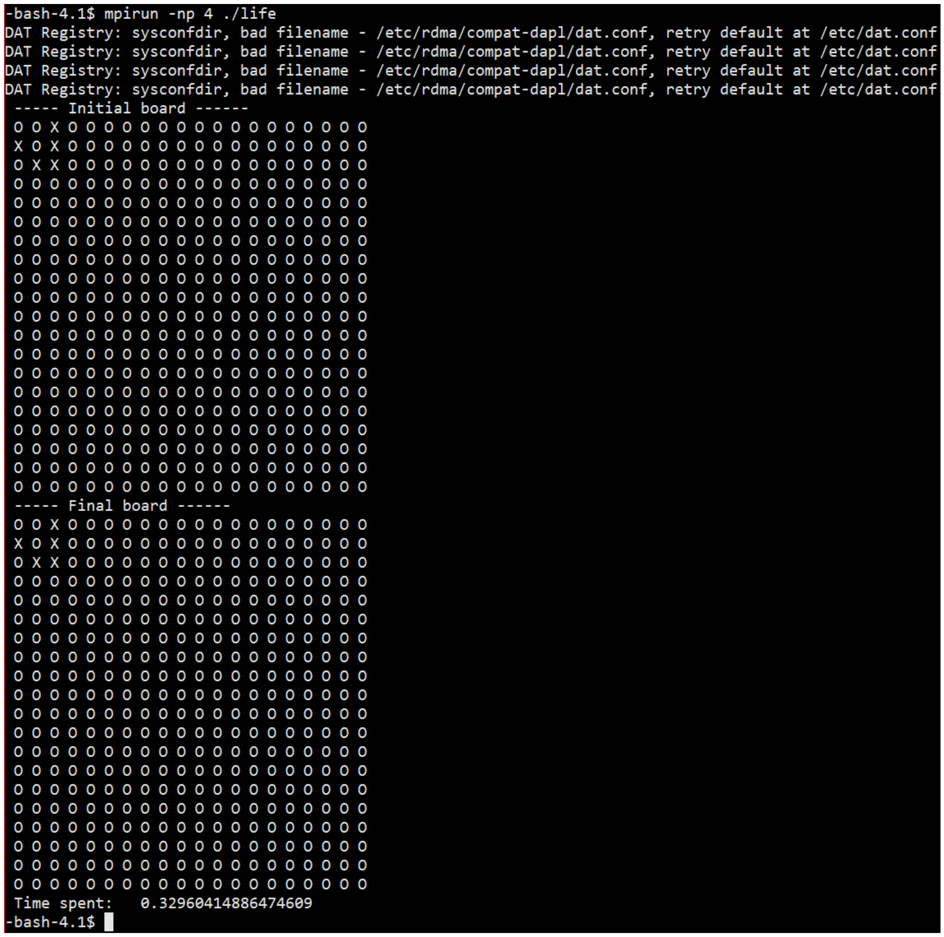
2 Documentation

For the full detail of the specifications, please refer to the **README.md** that came alone with the project.

3 Proof of Work

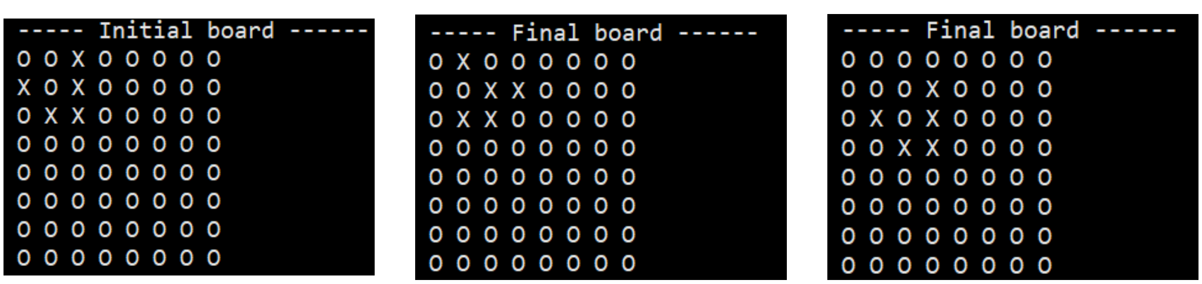
To prove that my GoF works correctly, I setup a 20x20 board, with the initial pattern look like in Figure 1. I ran 80 steps, and the resulting pattern has returned to its starting locations as shown in the Figure. Also, in Figure 2, I have shown the result of running 1 step and 4 step. Notice at 4 step that this returns back to the same pattern but with everything shifted diagonally by 1 square.

For the parrallel IO, I used *MPI_FILE_WRITE* to allow each processor to write to the same file in parrell. To do so I have to reserve bytes in the file, and allocate each rank with its own region in the file so that there is no when conflict. I have logged the state of the GoF simulation at each of the following step: $\{0, 19, 39, 59, 79\}$. Therefore, in the file you can see what each processors contains.



```
-bash-4.1$ mpirun -np 4 ./life
DAT Registry: sysconfdir, bad filename - /etc/rdma/compat-dapl/dat.conf, retry default at /etc/dat.conf
DAT Registry: sysconfdir, bad filename - /etc/rdma/compat-dapl/dat.conf, retry default at /etc/dat.conf
DAT Registry: sysconfdir, bad filename - /etc/rdma/compat-dapl/dat.conf, retry default at /etc/dat.conf
DAT Registry: sysconfdir, bad filename - /etc/rdma/compat-dapl/dat.conf, retry default at /etc/dat.conf
 ----- Initial board ------
 O O X O O O O O O O O O O O O O O O O O
 X O X O O O O O O O O O O O O O O O O O
 O X X O O O O O O O O O O O O O O O O O
 O O O O O O O O O O O O O O O O O O O O
 O O O O O O O O O O O O O O O O O O O O
 O O O O O O O O O O O O O O O O O O O O
 O O O O O O O O O O O O O O O O O O O O
 O O O O O O O O O O O O O O O O O O O O
 O O O O O O O O O O O O O O O O O O O O
 O O O O O O O O O O O O O O O O O O O O
 O O O O O O O O O O O O O O O O O O O O
 O O O O O O O O O O O O O O O O O O O O
 O O O O O O O O O O O O O O O O O O O O
 O O O O O O O O O O O O O O O O O O O O
 O O O O O O O O O O O O O O O O O O O O
 O O O O O O O O O O O O O O O O O O O O
 O O O O O O O O O O O O O O O O O O O O
 O O O O O O O O O O O O O O O O O O O O
 O O O O O O O O O O O O O O O O O O O O
 O O O O O O O O O O O O O O O O O O O O
 ----- Final board ------
 O O X O O O O O O O O O O O O O O O O O
 X O X O O O O O O O O O O O O O O O O O
 O X X O O O O O O O O O O O O O O O O O
 O O O O O O O O O O O O O O O O O O O O
 O O O O O O O O O O O O O O O O O O O O
 O O O O O O O O O O O O O O O O O O O O
 O O O O O O O O O O O O O O O O O O O O
 O O O O O O O O O O O O O O O O O O O O
 O O O O O O O O O O O O O O O O O O O O
 O O O O O O O O O O O O O O O O O O O O
 O O O O O O O O O O O O O O O O O O O O
 O O O O O O O O O O O O O O O O O O O O
 O O O O O O O O O O O O O O O O O O O O
 O O O O O O O O O O O O O O O O O O O O
 O O O O O O O O O O O O O O O O O O O O
 O O O O O O O O O O O O O O O O O O O O
 O O O O O O O O O O O O O O O O O O O O
 O O O O O O O O O O O O O O O O O O O O
 O O O O O O O O O O O O O O O O O O O O
 O O O O O O O O O O O O O O O O O O O O
 Time spent:   0.32960414886474609
-bash-4.1$
```

Figure 1: Output of the GoF simulated 80 steps on a 20x20 grid using 4 processors. The pattern should return to where it started. This output was printed on main thread with *print*.



```
----- Initial board ------
O O X O O O O O
X O X O O O O O
O X X O O O O O
O O O O O O O O
O O O O O O O O
O O O O O O O O
O O O O O O O O
O O O O O O O O
```

(a)

```
----- Final board ------
O X O O O O O O
O O X X O O O O
O X X O O O O O
O O O O O O O O
O O O O O O O O
O O O O O O O O
O O O O O O O O
O O O O O O O O
```

(b)

```
----- Final board ------
O O O O O O O O
O O O X O O O O
O X O X O O O O
O O X X O O O O
O O O O O O O O
O O O O O O O O
O O O O O O O O
O O O O O O O O
```

(c)

Figure 2: (a) The initial 8x8 board starting pattern. (b) the pattern after 1 step simulation. (c) the pattern after 4 step simulation. As shown above, the GoF was simulated correctly on any number of processes.

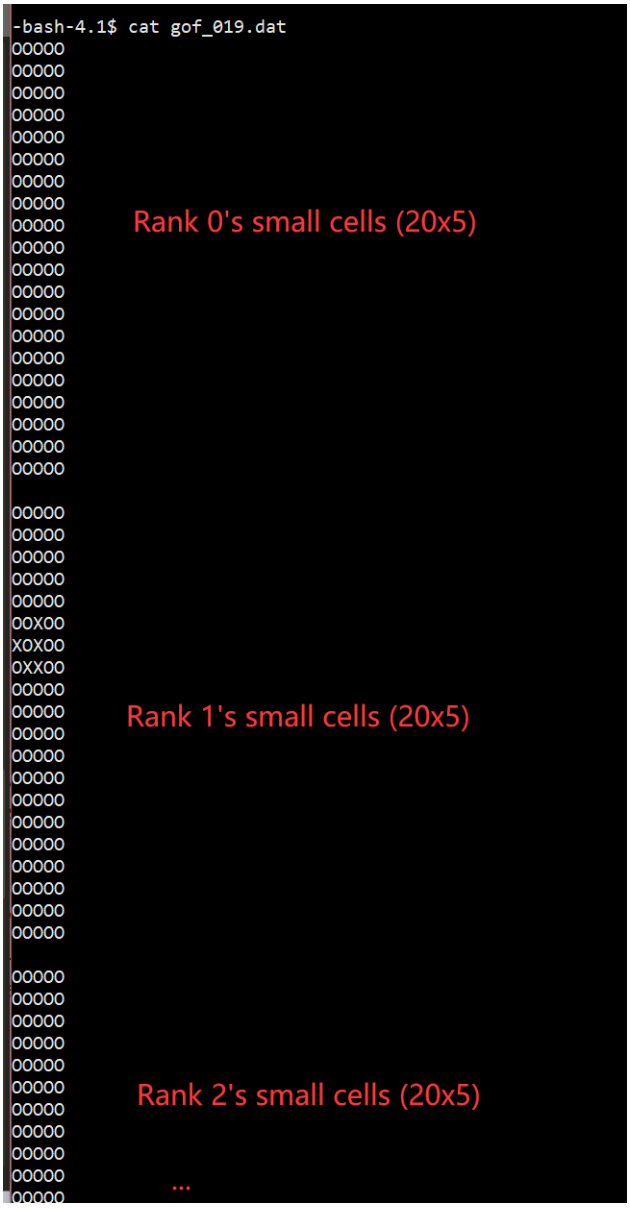


Figure 3: Parallel I/O. The state of each smaller cells were written into files (*gof_000.dat, gof_019.dat, gof_039.dat, gof_059.dat, gof_079.dat*). Example shows the state of a 20x20 cells that was running on 4 ranks of 1D decomposition, each rank has 20x5 smaller cells.

		10x10 (100 cells)					
Num Processors / board size		try 1	try 2	try 3	try 4	try 5	Average
1		3.23E-03	3.04E-03	3.23E-03	2.94E-03	3.38E-03	3.16E-03
2		5.71E-04	5.94E-04	5.90E-04	6.16E-04	5.89E-04	5.92E-04
4							#DIV/0!
5		1.76E-03	1.23E-03	1.32E-03	1.35E-03	3.72E-03	1.88E-03
10		3.84E-03	3.98E-03	4.39E-03	3.80E-03	4.02E-03	4.01E-03
20							#DIV/0!
		100x100 (10000 cells)					
Num Processors / board size		try 1	try 2	try 3	try 4	try 5	Average
1		4.58E-03	4.63E-03	3.97E-03	3.74E-03	3.67E-03	4.12E-03
2		9.36E-04	9.30E-04	9.24E-04	9.44E-04	1.02E-03	9.50E-04
4		1.10E-03	1.28E-03	1.21E-03	1.10E-03	1.11E-03	1.16E-03
5		1.59E-03	1.55E-03	1.43E-03	1.42E-03	1.68E-03	1.53E-03
10		4.15E-03	4.25E-03	5.28E-03	3.86E-03	3.80E-03	4.27E-03
20		2.06E-02	3.15E-02	2.05E-02	1.69E-02	2.45E-02	2.28E-02
		100x1000 (100000 cells)					
Num Processors / board size		try 1	try 2	try 3	try 4	try 5	Average
1		2.91E-01	2.90E-01	2.90E-01	2.90E-01	2.91E-01	2.90E-01
2		1.66E-01	1.45E-01	1.45E-01	1.45E-01	1.45E-01	1.49E-01
4		7.42E-02	7.49E-02	7.41E-02	7.44E-02	7.42E-02	7.44E-02
5		5.91E-02	5.95E-02	5.88E-02	5.90E-02	6.56E-02	6.04E-02
10		5.90E-02	5.74E-02	5.78E-02	5.76E-02	5.77E-02	5.79E-02
20		5.06E-02	5.10E-02	5.22E-02	4.82E-02	5.07E-02	5.06E-02
		1000x1000 (1000000 cells)					
Num Processors / board size		try 1	try 2	try 3	try 4	try 5	Average
1		3.07E+00	3.08E+00	3.09E+00	3.08E+00	3.08E+00	3.08E+00
2		1.44E+00	1.44E+00	1.44E+00	1.59E+00	1.44E+00	1.47E+00
4		8.35E-01	7.93E-01	7.93E-01	8.24E-01	7.92E-01	8.07E-01
5		6.46E-01	6.68E-01	6.44E-01	6.52E-01	6.45E-01	6.51E-01
10		5.28E-01	5.15E-01	5.95E-01	5.15E-01	5.25E-01	5.36E-01
20		4.41E-01	4.84E-01	4.55E-01	4.75E-01	4.91E-01	4.69E-01

(a)

	100x100 (10000 cells)					
Num Processors / board size	try 1	try 2	try 3	try 4	try 5	Average
1	4.28E-03	4.19E-03	4.23E-03	5.17E-03	4.29E-03	4.43E-03
4	2.39E-03	1.36E-03	1.52E-03	1.23E-03	4.34E-03	2.17E-03
25	1.63E-02	2.90E-02	1.08E-02	2.91E-02	9.88E-03	1.90E-02
	3600x3600 (12960000 cells)					
Num Processors / board size	try 1	try 2	try 3	try 4	try 5	Average
1	2.21E+00	2.21E+00	2.21E+00	2.21E+00	2.21E+00	2.21E+00
4	1.48E-01	1.40E-01	1.56E-01	1.40E-01	1.39E-01	1.45E-01
9	7.59E-02	8.41E-02	8.74E-02	8.51E-02	8.10E-02	8.27E-02
16	4.11E-02	4.30E-02	3.13E-02	2.87E-02	3.16E-02	3.51E-02
25	2.65E-02	3.33E-02	5.12E-02	3.73E-02	2.59E-02	3.48E-02
	100x1000 (100000 cells)					
Num Processors / board size	try 1	try 2	try 3	try 4	try 5	Average
1	1.78E-02	1.77E-02	1.77E-02	1.78E-02	1.80E-02	1.78E-02
4	2.54E-03	2.43E-03	2.57E-03	2.35E-03	2.37E-03	2.45E-03
25	2.67E-02	2.51E-02	3.01E-02	1.98E-02	1.28E-02	2.29E-02
	720x720 (518400 cells)					
Num Processors / board size	try 1	try 2	try 3	try 4	try 5	Average
1	8.00E-02	7.98E-02	8.30E-02	8.25E-02	8.26E-02	8.16E-02
4	6.07E-03	6.46E-03	6.46E-03	6.87E-03	8.42E-03	6.86E-03
9	7.65E-03	5.30E-03	5.54E-03	5.00E-03	5.76E-03	5.85E-03
16	1.24E-02	1.37E-02	1.00E-02	9.21E-03	1.60E-02	1.23E-02

(b)

Figure 4: (a) Time measured from 1D decomposition (b) Time measured from 2D decomposition.

4 Performance Model

Say we have number of grid N_y, N_x

Say t_c is the computation time of single cell. By looking at Figure 4, we can compute the $t_c \approx 3.16E-05$ by dividing the average computation time with the number of cells. This is going to be our baseline, we assume the computation time of each cell is the same.

4.1 1D Decomposition

Assume 1-D domain decomposition: partitioned into columns $(N_y \times 1)$.

P tasks for sub-grids $\frac{N}{P} \times \frac{N}{P} \times N_z$

I have run the code on 10x10, 100x100, 100x1000, 1000x1000 cells with 1, 2, 4, 5, 10, 20 number of processors, the results are shown in Figure 4. For each test I have ran 5 times, and took the average to get a more accurate result (Shown in Green box).

Immediately we notice that this decompositon strategy does not work that well on small board size. But on a larger board size (like the 1000x1000), the amount of time to complete was significantly reduced. This is because of the communication overhead on smaller boards does not trade well the efficiency.

4.2 2D Decomposition

Assume 2-D domain decomposition: partitioned into tiles $(N_y \times N_x)$.