MPI Project: Game of life

Tanuj Gupta

1. PCAM process

Summary:

I have implemented the Giga version(level 6) that decomposes the matrix in both directions.

1.1 Partitioning

- Partitioning involves decomposing computation and data into small tasks.

- To implement the Giga version and achieve maximum parallelism, I have decomposed the N × N grid in both directions. In other words, I have decomposed N × N grid into 2D subgrids

- By partitioning the N × N grid into 2D subgrids, we can assign a separate task to all these subgrids that can concurrently and independently solve the game of life.

1.2 Communication

We have decided to decompose the N × N grid into 2D subgrids. To be able to update the nodes on the edges, these sub grids need information from other nodes.

So every of these 2D subgrids would communicate to and from 8 neighbours:
A) Left and right Columns communication with 2 neigbours on the right and left

Edge cases: Ring communication
The subgrids on the left edge will communicate their left columns with the last subgrid in the same row. The subgrids on the right edge will communicate their right columns with the he subgrids on the left edge will communicate their left

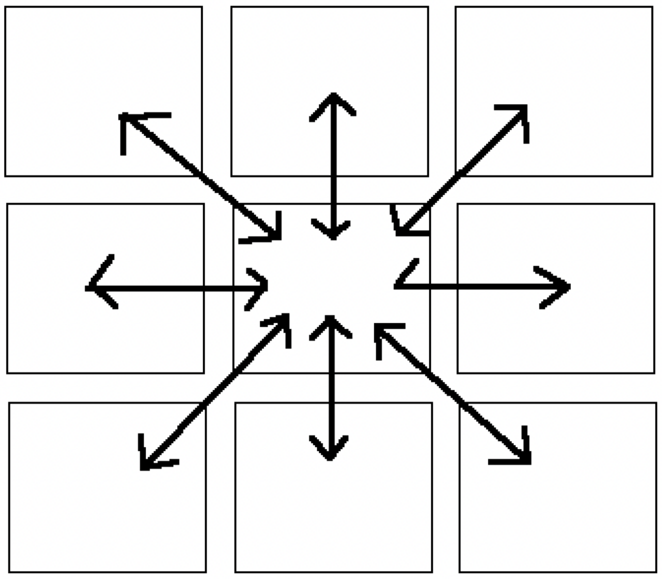
columns with

B) Top and bottom Rows communication with 2 neigbours on the bottom and Top

Edge cases: Ring communication

The subgrids on the top edge will communicate their top row with the last subgrid in the same column. The subgrids on the bottom edge will communicate their bottom row with the first subgrid in the same column

C) 4 corners communication with 4 neigbours on the diagonals.



Code Implementation for communication:

- Function calculate_neighbour_nodes in utility_func.f90 calculates all the 8 neigbour subgrids for a given subgrid
- Function communicate_columns_to_from_neigbours in utility_func.f90 communicates column information to and fro from 2 neigbours up and down
- Function communicate_rows_to_from_neigbours in utility_func.f90 communicates row information to and fro from 2 neigbours left and right
- Function communicate_diagnols_to_from_neigbours in utility_func.f90 communicates corner information to and fro from 4 neigbour on the diagnals

Other Communications:

1. Scatter subgrids to all nodes from nodes 0
Initially, the game data will be built on Node with rank 0. This node will have to scatter corresponding subgrids data to all the other nodes.

> **Code Implementation:**
> Functions send_initial_grid_to_nodes and recieve_initial_grid_to_nodes in utility_func.f90 carry this communication.

2. Gather and aggregate updates on node 0 from all nodes
After update of the game locally on the 2D subgrid, all subgrids will send the update back to with rank 0.

> **Code Implementation:**
> Functions send_updated_grid_to_pid0 and recieve_updated_grid_to_pid0 in utility_func.f90 carry this communication

1.3 Agglomeration

- We are decomposing the N × N grid into 2D subgrids. Theoretically as per partitioning, these subgrids can be as small as 1 × 1 size. Every subgrid will communicate with 8 of its neigbours. If we divide N × N grid into M such subgrids, total communication will be 8*M.
- To reduce the amount of communication, we can group these smaller subgrids into larger ones. Larger size subgrids reduces communication costs and allows for tasks to be executed without interruption

1.4 Mapping

To perfectly balance the tasks across p processes, we need to assign equal sized grids to all processors. In an ideal scenario, every processor will get 2D subgrid of size $\frac{(N \times N)}{P} = \frac{N}{\sqrt{P}} \times \frac{N}{\sqrt{P}}$. That is, we can assign each processor a square subgrid with dimension $\frac{N}{\sqrt{p}}$. But this is possible only if:

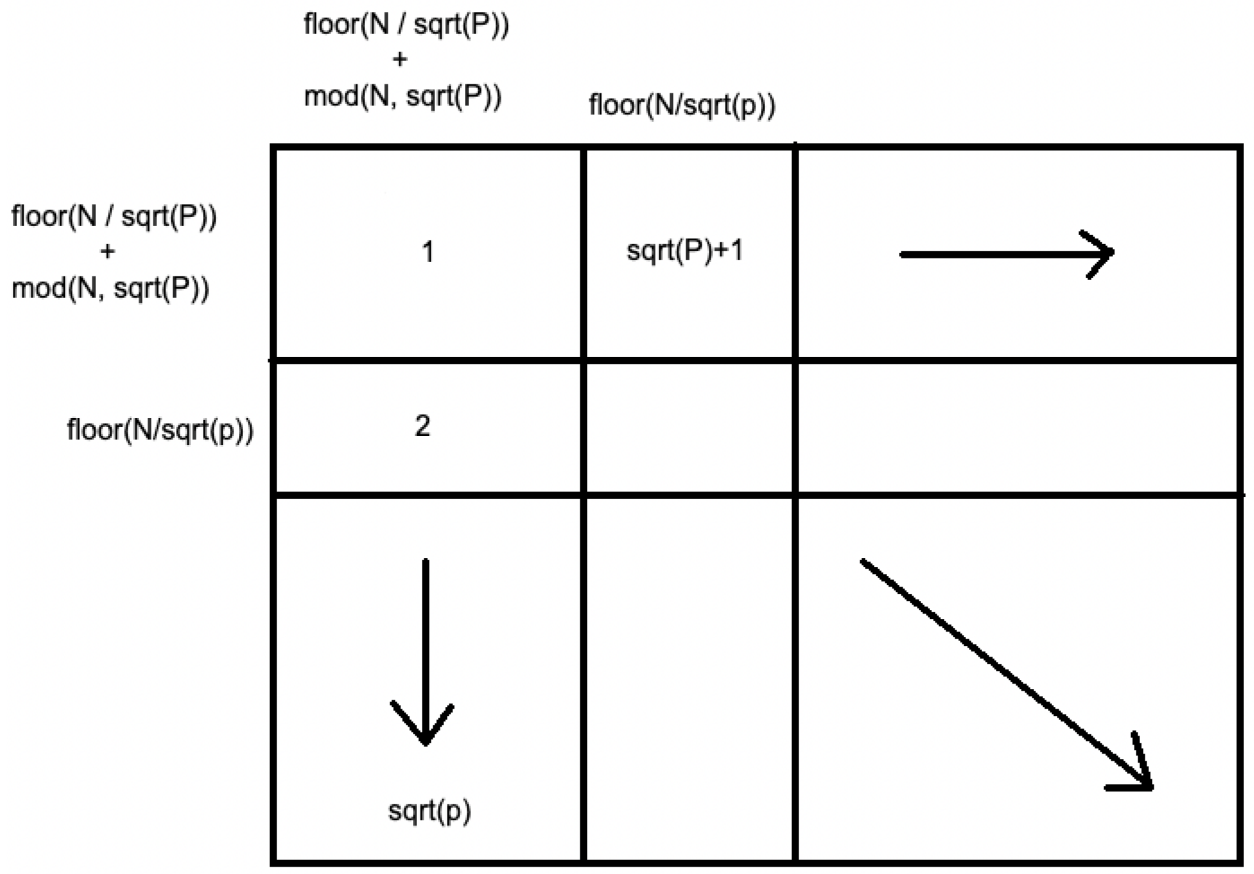
1. P **is a perfect square.**

To handle a more general scenario, we in our code factorize P into X and Y such that P=X $\times$ Y. To make the division the most load balanced, we take X,Y such that |X - Y| is minimum.

2. $\sqrt{P}$ **is divisible by N**

To handle a more general scenario, where $\sqrt{P}$ is not divisible by N, we assign the excess to the processors in first row and first column. In other words:

- The very first processor will be assigned a 2D subgrid of size $(floor(\frac{N}{\sqrt{P}}) + mod(N,P)) \times (floor(\frac{N}{\sqrt{P}}) + mod(N,P))$
- Processors in the first row will be assigned a 2D subgrid of size $(floor(\frac{N}{\sqrt{P}}) + mod(N,P)) \times (\frac{N}{\sqrt{P}})$
- Processors in the first column will be assigned a 2D subgrid of size $(floor(\frac{N}{\sqrt{P}})) \times (floor(\frac{N}{\sqrt{P}}) + mod(N,P))$
- All the other processors will be assigned a 2D subgrid of size$floor(\frac{N}{\sqrt{P}}) \times floor(\frac{N}{\sqrt{P}})$



Code Implementation for mapping:
- Function calculate_local_grid_position in utility_func.f90 calculates relative position of processor on the grid in terms of row/col. Like processor 1 will have position 1,1. Processor P **will have position** $\sqrt{P}, \sqrt{P}$
- Function calculate_local_grid_size in utility_func.f90 calculates the size of the 2D subgrid assigned to processor number n
- Function calculate_local_grid_start_position in utility_func.f90 calculates the indexes of first cell of the 2D subgrid assigned to processor number n.

2. How to run the code

On grape, do the following to compile and run my code:

STEP 1: Make sure all the required libraries are loaded using commands:
1. module load rocks-openmpi
2. module load gcc-6.2.0

STEP 2: To change the
A) Grid size: Edit game_of_life_mpi_mega_version.f90 and change variable grid_size
B) The number of processors: Edit game_of_life_mpi_mega_version.f90 and and change variable num_procs
C) The number of iterations: Edit game_of_life_mpi_mega_version.f90 and change variable num_of_iterations

By default, values of these variables are:
grid_size=20
num_procs=4
num_of_iterations=80

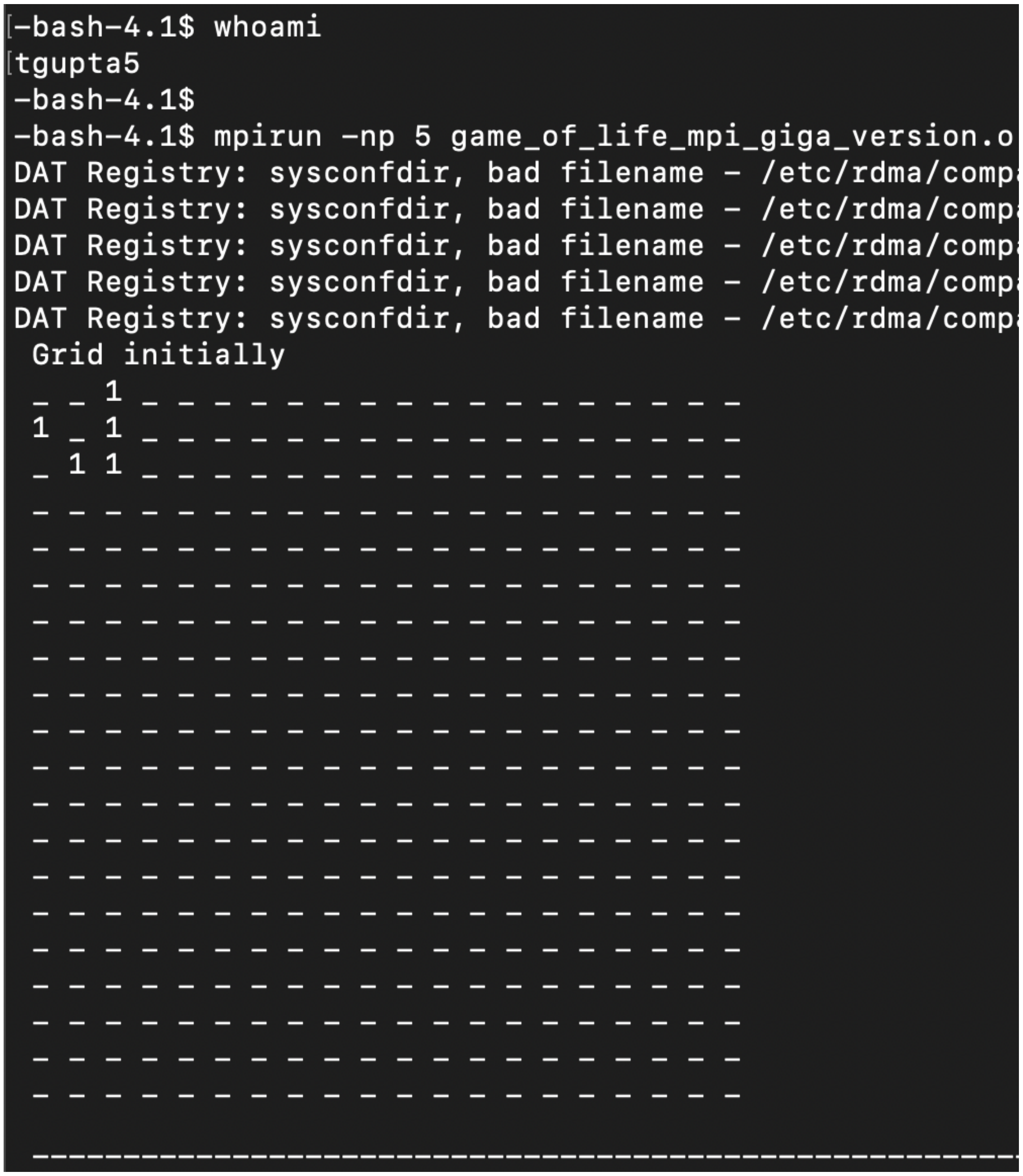
STEP 3: To compile my code, run the following commands in order

```
mpif90 -o utility_func.o -c  utility_func.f90
mpif90 -o game_of_life_mpi_giga_version.o  utility_func.o  game_of_life_mpi_giga_version.f90
```

STEP 4: To run my code with n processors, run the following command

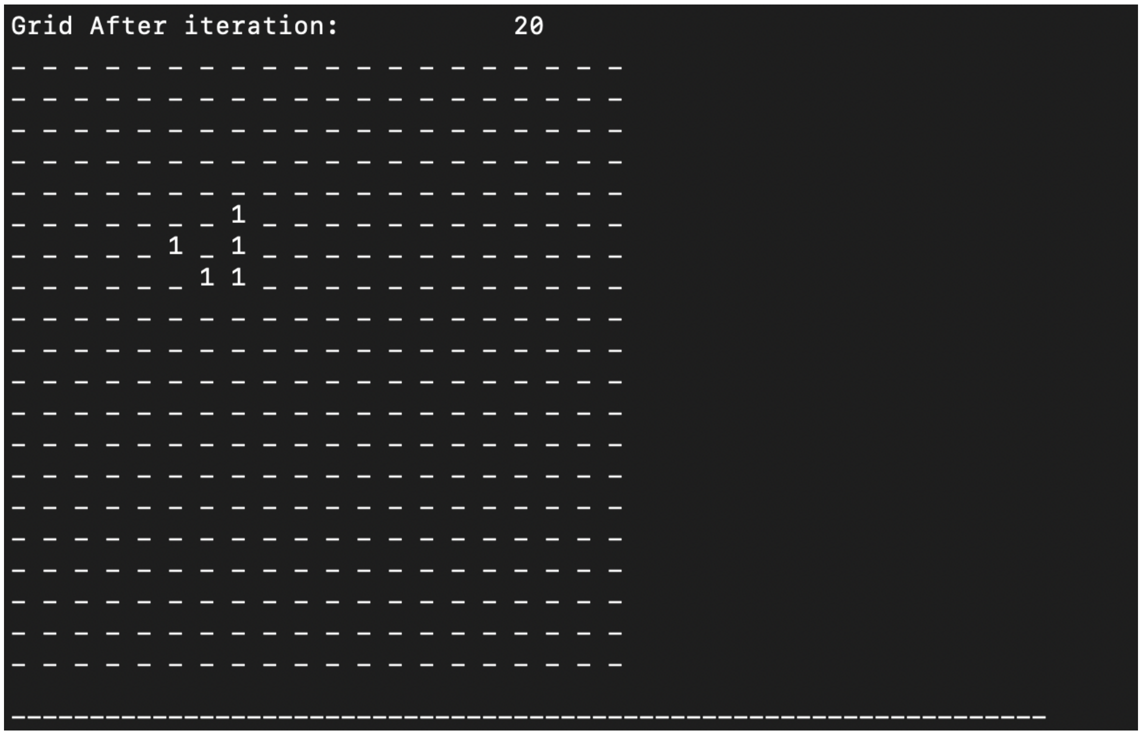
```
mpirun -np n game_of_life_mpi_giga_version.o
```

3 Code output for 20×20 Grid



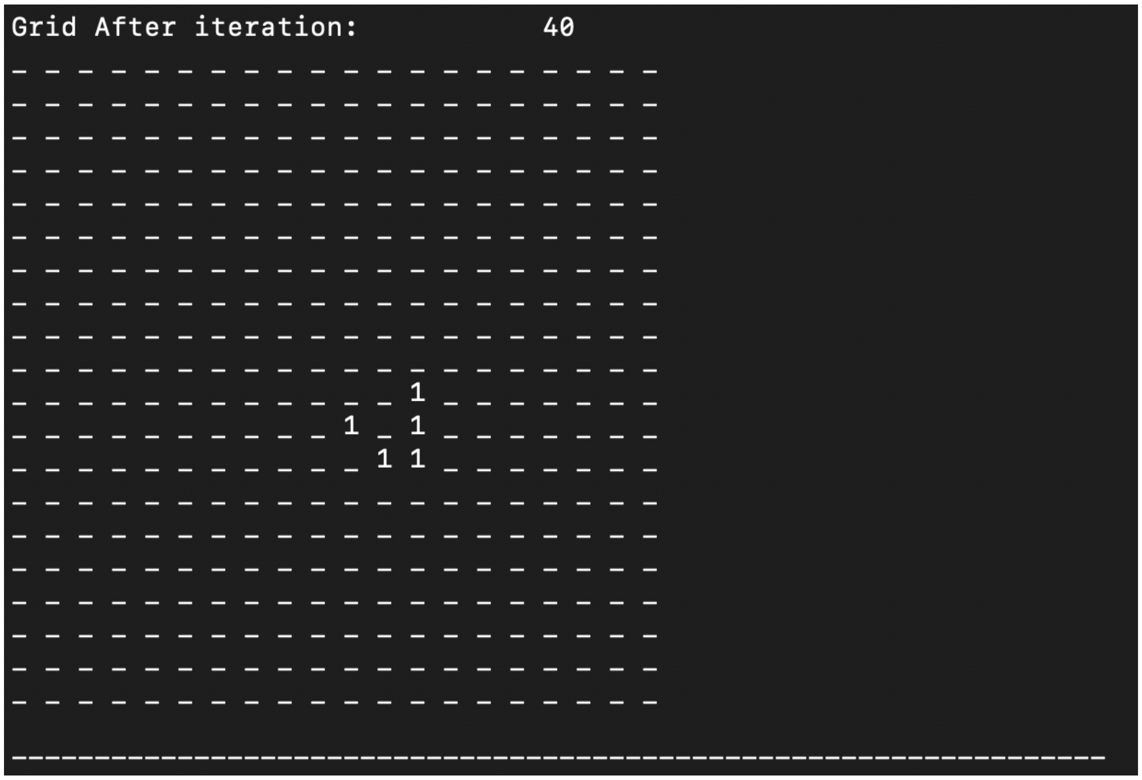
```
[-bash-4.1$ whoami
[tgupta5
 -bash-4.1$
 -bash-4.1$ mpirun -np 5 game_of_life_mpi_giga_version.o
 DAT Registry: sysconfdir, bad filename - /etc/rdma/comp
 DAT Registry: sysconfdir, bad filename - /etc/rdma/comp
 DAT Registry: sysconfdir, bad filename - /etc/rdma/comp
 DAT Registry: sysconfdir, bad filename - /etc/rdma/comp
 DAT Registry: sysconfdir, bad filename - /etc/rdma/comp
  Grid initially
  _ _ 1 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
  1 _ 1 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
  _ 1 1 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

  ------------------------------------------------------
```



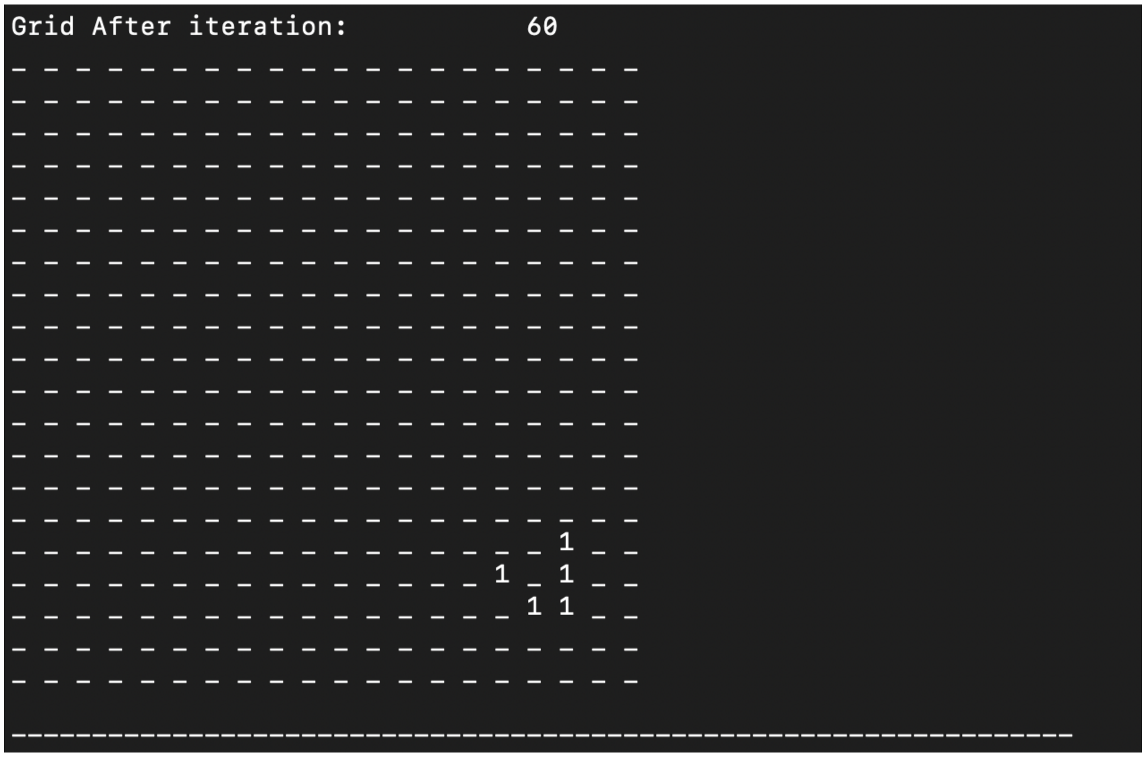
```
Grid After iteration:           20
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ 1 _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ 1 _ 1 _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ 1 1 _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

------------------------------------------------------------------
```



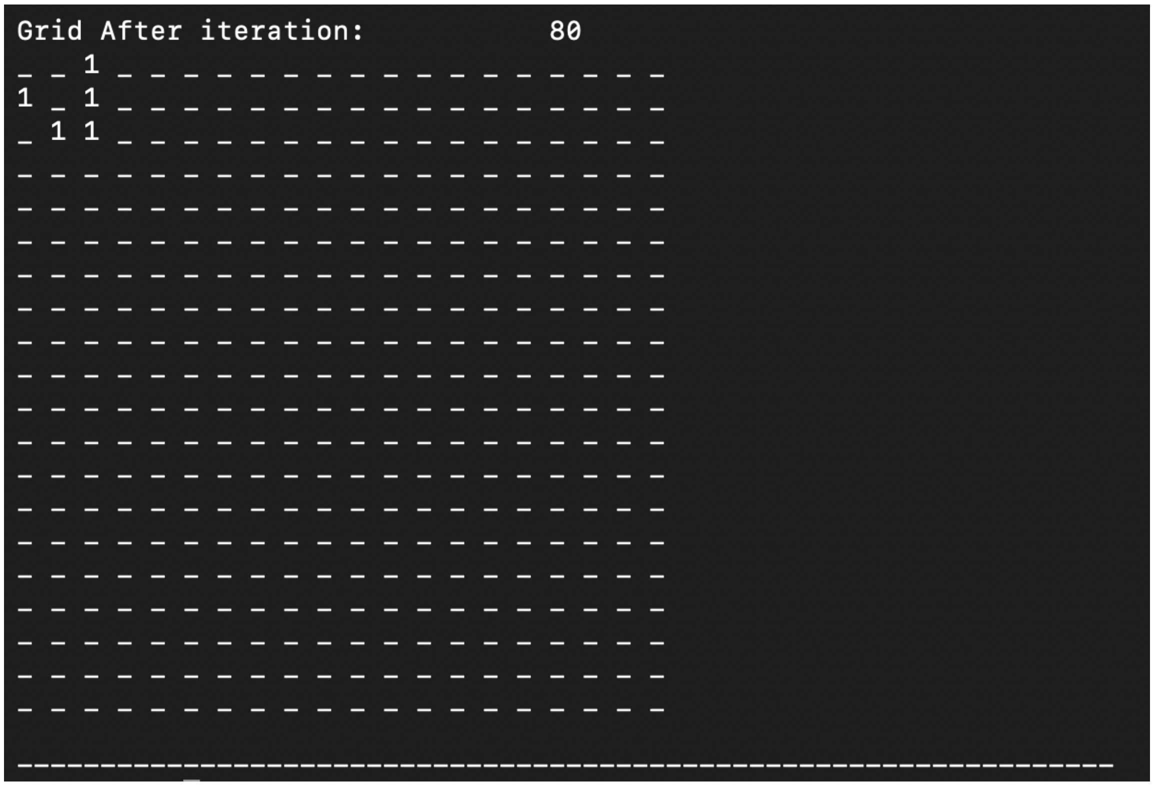
```
Grid After iteration:           40
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ 1 _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ 1 _ 1 _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ 1 1 _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

------------------------------------------------------------------
```



```
Grid After iteration:           60
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ 1 _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ 1 _ 1 _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ 1 1 _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

------------------------------------------------------------------
```



```
Grid After iteration:           80
_ _ 1 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
1 _ 1 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ 1 1 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

------------------------------------------------------------------
```

4. Performance Model and Testing

Assume
- **Dimentions =** $Nx \times Ny$, where Nx **=** Ny **=** N for simplicity
- **Number of processes** = *P*
- 2-D domain decomposition

Given the dimensions:

$$Total\ grid\ points\ =\ N \times N\ = N^2 \qquad -\ (1)$$

Computation time

Assuming No replicated computation:

$$Tcomp\ =\ T_c \times N^2 \qquad -\ (2)$$

where t_c = average computation time per grid point (slightly different at edges from interior etc)

Communication time

Given that its a 2-D domain decomposition, we will partition the grid points among P processors in 2 directions, namely, X and Y.
Assuming that the partitioning is equal in both the directions, **the dimension of task per processor** would then be:

X direction: $\frac{N}{\sqrt{P}}$, Y direction: $\frac{N}{\sqrt{P}}$, Z direction: N_Z

$$Total\ grid\ points\ per\ processor\ =\ \frac{N}{\sqrt{P}} \times \frac{N}{\sqrt{P}} \qquad -\ (3)$$

Given that its a 2d decomposition, each task communicates with 8 neighbours. With 4 of its neigbours, the size of the wall shared is $\frac{N}{\sqrt{P}}$. With 4 diagonal neigbours, the size of the wall shared is 1.

Therefore:

$$T_{comm}\ =\ 4P * (T_S + T_W * \frac{N}{\sqrt{P}})\ +\ 4P * (T_S + T_W)$$

$$\Rightarrow T_{comm}\ =\ 4P * (T_S + T_W * (\frac{N}{\sqrt{P}} + 1)) \qquad -\ (4)$$

4.1. Performance Model: 2D Game of life Total time

Assuming that P divides N exactly, then assume load-balanced and no idle time:

$$T_{2D\_gol} = (T_{comp} + T_{comm})/P$$

$$\Rightarrow T_{2D\_gol} = \frac{T_c N^2}{P} + 4P * \left(\frac{T_S + T_W * (\frac{N}{\sqrt{P}} + 1)}{P}\right)$$

$$\Rightarrow T_{2D\_gol} = \frac{T_c N^2}{P} + 4(T_S + T_W * (\frac{N}{\sqrt{P}} + 1))$$

$$\Rightarrow T_{2D\_gol} = \frac{T_c N^2}{P} + 4T_S + 4T_W * (\frac{N}{\sqrt{P}} + 1) \qquad - (5)$$

4.2. Values for T_s, T_w, and T_c

Values for T_s, T_w

From my HW6, values obtained for T_s and T_w were:

T_s = 0.007997999 sec

T_w = 0.0005823312 sec

Values for T_c

To calculate value for constant T_c, I used ones.f90 code used for HW2.

Basically

$$Tcomp = T_c \times N^2$$

$$\Rightarrow T_c = \frac{Tcomp}{N^2} \qquad -(6)$$

Using eq(6), we can estimate T_c. Idea is to compute (T_{comp}/N^2) for different values of N to estimate T_c. Then take average of these values to get accurate estimation of T_c

Data Point	**Value of N**	**Value of T_{comp}**	**Value of $T_c = T_{comp}/N^2$**
1	**500**	**0.0339939**	**$1.359759 * 10^{-7}$**
2	**1000**	**0.132979**	**$1.32979 * 10^{-7}$**

3	**3000**	**1.1698219**	**1.299802 * 10^{-7}**
4	**6000**	**4.6652909**	**1.29591416 * 10^{-7}**
5	**9000**	**10.4964039**	**1.295852345679 * 10^{-7}**
6	**15000**	**29.159567**	**1.29598075 * 10^{-7}**
Average -> $(\sum_{n=1}^{6} T_c)/6$			**1.31284970928 * 10^{-7}**

$$\Rightarrow T_c = 1.31284970928 * 10^{-7}$$

$$\Rightarrow T_s = 7.997999 * 10^{-3}$$

$$\Rightarrow T_w = 5.823312 * 10^{-4}$$

4.3. Testing

D.1 Time estimation using Performance model given by equation 5

For data points:
N= 20, 100, 500, 1000, 3000, 6000

As per equation 5

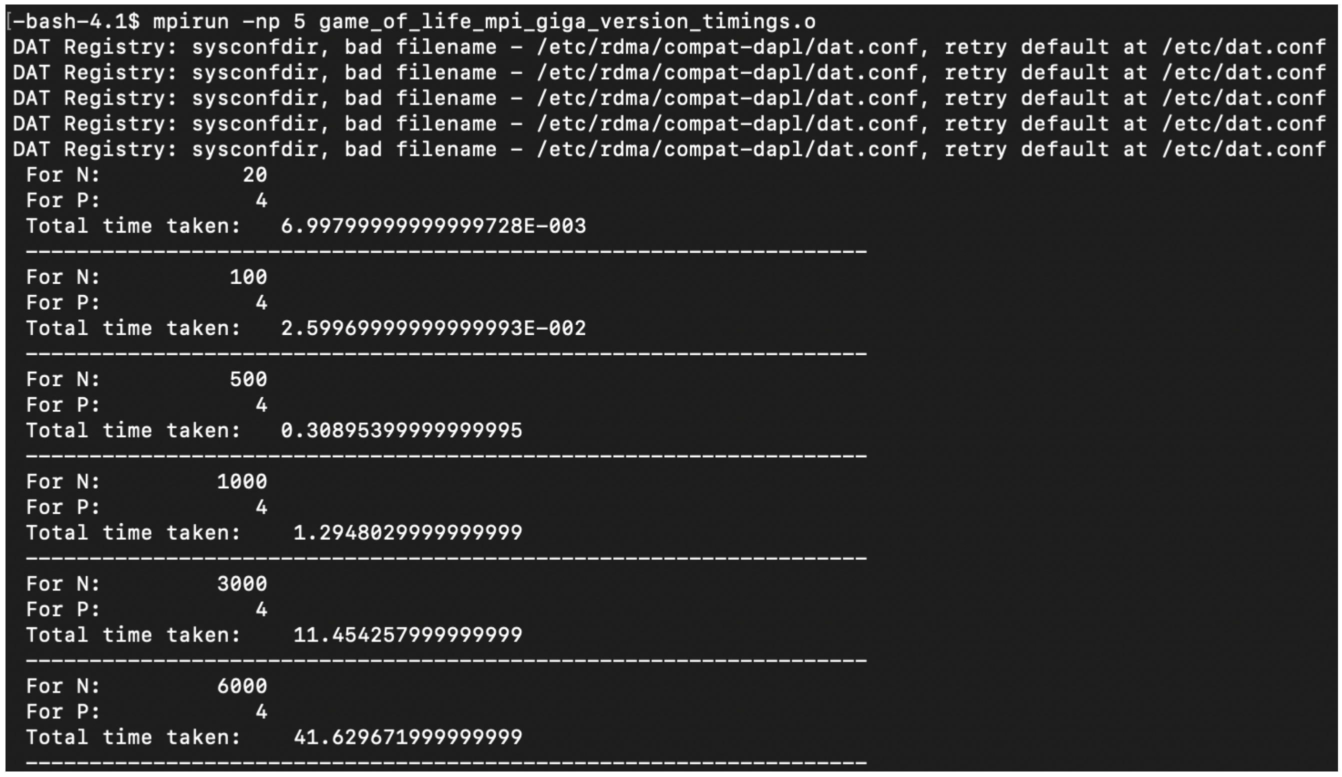
$$\Rightarrow T_{2D\_gol} = \frac{T_c N^2}{P} + 4T_S + 4T_W * (\frac{N}{\sqrt{P}} + 1)$$

Using
$P = 4,$
$T_c = 1.31284970928 * 10^{-7},$
$T_s = 7.997999 * 10^{-3},$
$T_W = 5.823312 * 10^{-5}$

Data Point	Value of N	Value of T_{2D_gol}
1	20	0.0576276971187326126

2	100	0.15111577458446845
3	500	0.62485783733427525
4	1000	1.2318049520254135
5	3000	3.8236998692154884
6	6000	8.2038607969880104

D.2 Actual Running Time using the code



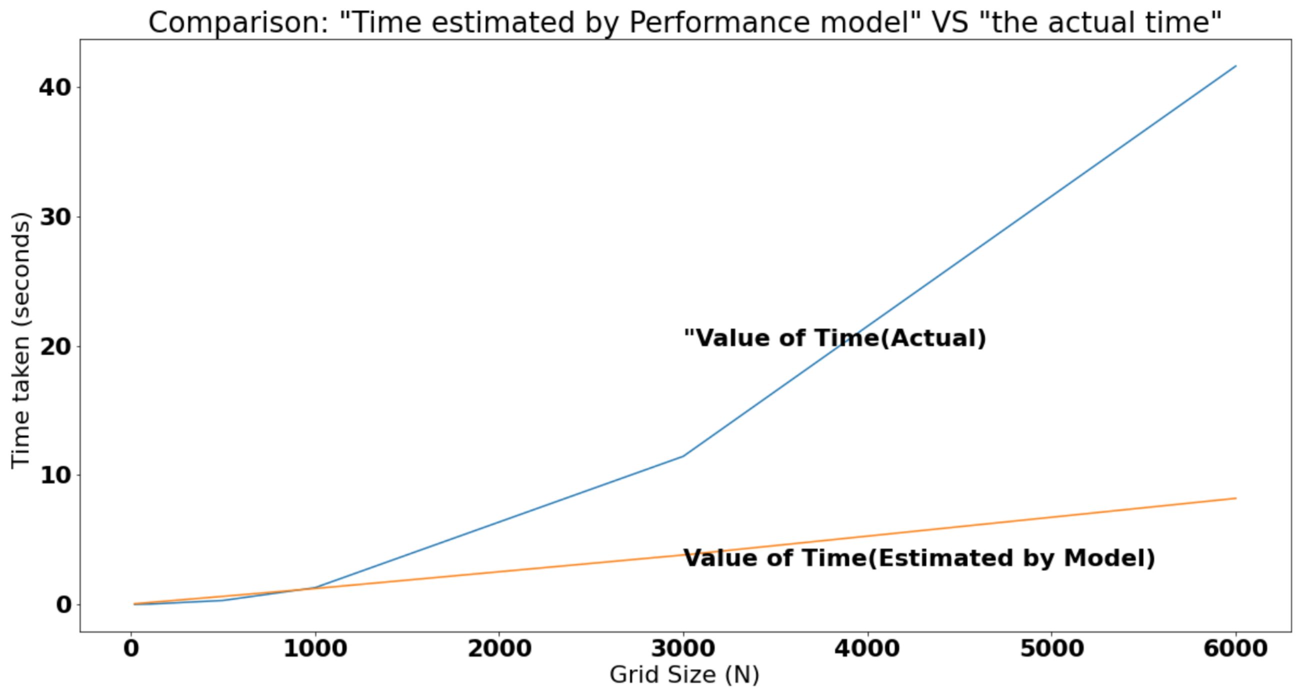
```
-bash-4.1$ mpirun -np 5 game_of_life_mpi_giga_version_timings.o
DAT Registry: sysconfdir, bad filename - /etc/rdma/compat-dapl/dat.conf, retry default at /etc/dat.conf
DAT Registry: sysconfdir, bad filename - /etc/rdma/compat-dapl/dat.conf, retry default at /etc/dat.conf
DAT Registry: sysconfdir, 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
DAT Registry: sysconfdir, bad filename - /etc/rdma/compat-dapl/dat.conf, retry default at /etc/dat.conf
 For N:           20
 For P:            4
 Total time taken:   6.99799999999999728E-003
 ------------------------------------------------------------------
 For N:          100
 For P:            4
 Total time taken:   2.59969999999999993E-002
 ------------------------------------------------------------------
 For N:          500
 For P:            4
 Total time taken:   0.30895399999999995
 ------------------------------------------------------------------
 For N:         1000
 For P:            4
 Total time taken:    1.2948029999999999
 ------------------------------------------------------------------
 For N:         3000
 For P:            4
 Total time taken:    11.454257999999999
 ------------------------------------------------------------------
 For N:         6000
 For P:            4
 Total time taken:    41.629671999999999
 ------------------------------------------------------------------
```

Data Point	Value of N	Value of T_{2D_gol}
1	20	0.00699799999999999728
2	100	0.0259969999999999993
3	500	0.30895399999999995
4	1000	1.2948029999999999
5	3000	11.454257999999999

6	6000	41.629671999999999

D.3 Comparison

Data Point	Value of N	Value of Time(Estimated by Model)	Value of Time(Actual)
1	20	0.0576276971187326126	0.0069979999999999728
2	100	0.15111577458446845	0.0259969999999999993
3	500	0.62485783733427525	0.308953999999999995
4	1000	1.2318049520254135	1.2948029999999999
5	3000	3.8236998692154884	11.454257999999999
6	6000	8.2038607969880104	41.629671999999999


Comparison: "Time estimated by Performance model" VS "the actual time"
Time taken (seconds)
Grid Size (N)
"Value of Time(Actual)
Value of Time(Estimated by Model)