Game of Life Report

1. PCAM

   Partitioning - We decompose this problem into smaller subdomains. To do that, my process involved decomposing the problem into smaller subdomains. We divide cell grid by columns
   Communication - I am using 4 processors. Also MPI to exchange data about ghost cells. Each processor handles a partitioned area
   Agglomeration - Each partitioned region is grouped into cells. I also use the gather MPI function to collect data from all 4 processes and put it together
   Mapping - Make sure that tasks are equally distributed to processors. In my 20 x 20, each of the 4 processors gets its fair share
2. My fortran code file is in gol.f90 and gol2.f90.
   To compile my file(s), write this command: mpif90 gol(2).f90 -o gol(2).
   To run my file, write this command: mpirun -np <Number of Processors> ./gol(2).
   To adjust the grid dimensions, modify nrows and ncols.
   To adjust the number of steps, adjust the value of N
   Also, you can uncomment the section of my code where I assign pseudo random boolean values to the grid.
   The difference between the 2 files is that the gol.f90 file is where I do question 3, and track the alive cells, whereas the gol2.f90 file is where I do column-wise domain decomposition.
   Below, I have snippets with comments explaining the purpose of each of those snippets:

```fortran
! Set the rows
game(1, :) = game(nrows + 1, :)
game(nrows + 2, :) = game(2, :)

! Set the columns
game(:, 1) = game(:, ncols + 1)
game(:, ncols + 2) = game(:, 2)

! Set the corners
game(1, 1) = game(nrows + 1, ncols + 1)
game(1, ncols + 2) = game(ncols + 1, 2)
game(nrows + 2, 1) = game(2, ncols + 1)
game(nrows + 2, ncols + 2) = game(2, 2)
```

```fortran
do i = 2, nrows + 1
    do j = 2, ncols + 1
        true_count = 0

        ! Checks all the cells around the current one
        do mini_i = i - 1, i + 1
            do mini_j = j - 1, j + 1
                if (mini_i /= i .or. mini_j /= j) then
                    if (game(mini_i, mini_j)) then
                        true_count = true_count + 1
                    end if
                end if
            end do
        end do

        ! Game of Life Rules
        if (true_count .eq. 3) then
            next_game(i, j) = .TRUE.
        else if (true_count .eq. 2) then
            continue
        else
            next_game(i, j) = .FALSE.
        end if
    end do
end do
```

```fortran
! Column-wise domain decomposition starts here
local_ncols = ncols / size

! Allocating memory for grids and communication buffers
! Each of the 4 processes will use its own grid
allocate(local_game(nrows + 2, local_ncols + 2))
allocate(local_next_game(nrows + 2, local_ncols + 2))
allocate(send_left(nrows + 2))
allocate(recv_left(nrows + 2))
allocate(send_right(nrows + 2))
allocate(recv_right(nrows + 2))
allocate(send_up(local_ncols + 2))
allocate(recv_up(local_ncols + 2))
allocate(send_down(local_ncols + 2))
allocate(recv_down(local_ncols + 2))
```

```fortran
! If there is > 1 processors, exchange info about grid ghost cells
if (size > 1) then
    send_left = local_game(:, 2)
    send_right = local_game(:, local_ncols + 1)
    send_up = local_game(2, :)
    send_down = local_game(nrows + 1, :)

    call MPI_SENDRECV(send_left, nrows + 2, MPI_LOGICAL, left, 1, &
                      recv_right, nrows + 2, MPI_LOGICAL, right, 1, &
                      MPI_COMM_WORLD, MPI_STATUS_IGNORE, ierr)

    call MPI_SENDRECV(send_right, nrows + 2, MPI_LOGICAL, right, 2, &
                      recv_left, nrows + 2, MPI_LOGICAL, left, 2, &
                      MPI_COMM_WORLD, MPI_STATUS_IGNORE, ierr)

    call MPI_SENDRECV(send_up, local_ncols + 2, MPI_LOGICAL, up, 3, &
                      recv_down, local_ncols + 2, MPI_LOGICAL, down, 3, &
                      MPI_COMM_WORLD, MPI_STATUS_IGNORE, ierr)

    call MPI_SENDRECV(send_down, local_ncols + 2, MPI_LOGICAL, down, 4, &
                      recv_up, local_ncols + 2, MPI_LOGICAL, up, 4, &
                      MPI_COMM_WORLD, MPI_STATUS_IGNORE, ierr)
    local_game(:, 1) = recv_left
    local_game(:, local_ncols + 2) = recv_right
    local_game(1, :) = recv_up
    local_game(nrows + 2, :) = recv_down
end if
```

3. Here are my grids at steps 0, 20, 40, and 80 below:

```
Initial Boolean Matrix
Initial No. of Cells Alive:           5
F F T F F F F F F F F F F F F F F F F F
T F T F F F F F F F F F F F F F F F F F
F T T F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
```

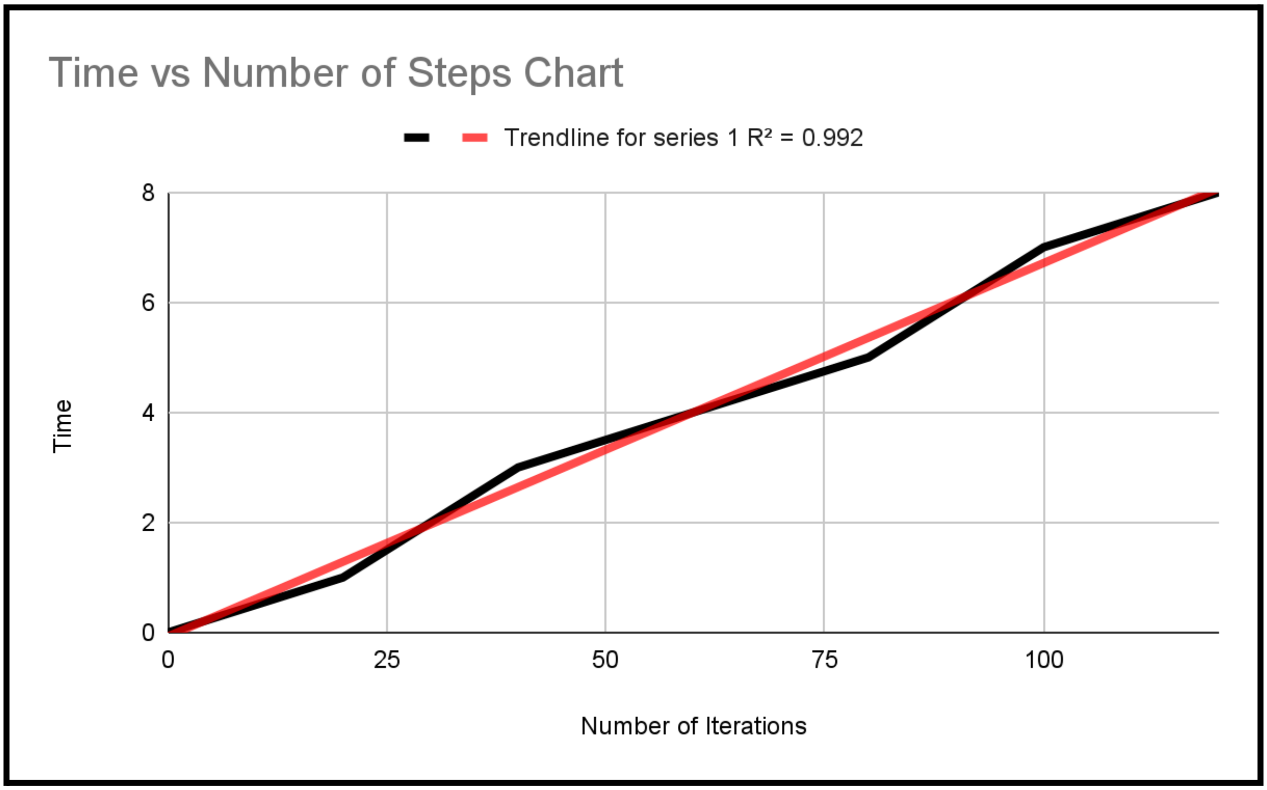
```
Grid After          20  steps
Final No. of Cells Alive:          5
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F T F F F F F F F F F F F F
F F F F F T F T F F F F F F F F F F F F
F F F F F F T T F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
```

```
Grid After          40  steps
Final No. of Cells Alive:          5
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F T F F F F F F F
F F F F F F F F F F T F T F F F F F F F
F F F F F F F F F F F T T F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
```

```
Grid After          80  steps
Final No. of Cells Alive:          5
F F T F F F F F F F F F F F F F F F F F
T F T F F F F F F F F F F F F F F F F F
F T T F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
F F F F F F F F F F F F F F F F F F F F
```

I get the same pattern every 4 steps. The full pattern moves down and right 1 for every 4 steps. After each step is completed, I immediately adjust the periodic boundary conditions which allows for this pattern to be repeated. Since it is a 20 x 20, it takes 80 (20 x 4) steps to reach the exact same grid again. I also used 4 processors to do this.

4. Here's my plot:



This graph shows that there is a strong linear correlation between the execution time and the number of iterations. I used system_clock instead and iterations in increments of 20. The trendline equation is: $Time = 0.0679 * No.\ of\ Iterations - 0.0714$. The 0.0679 is very close to 0.05, so it is very close to a linear rate of an increase of 1 in time for every increase in 20 iterations. Startup cost here would technically be 0 and the cost per iteration would be 0.0679.