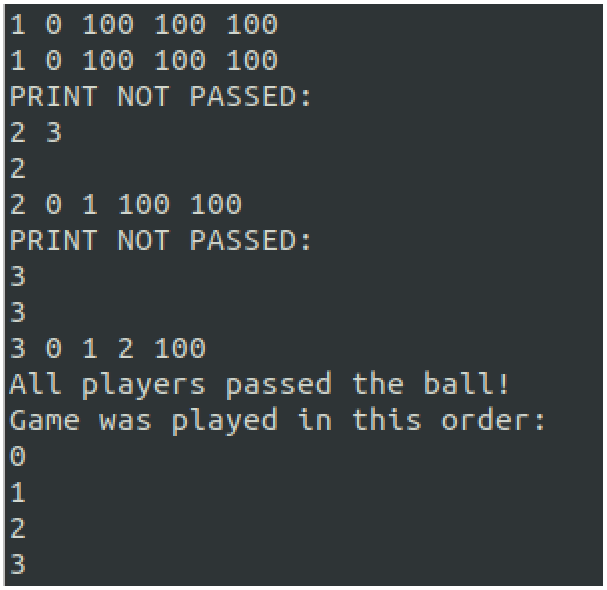
HW: MPI - Report

Ping–pong

To run the code do the following:

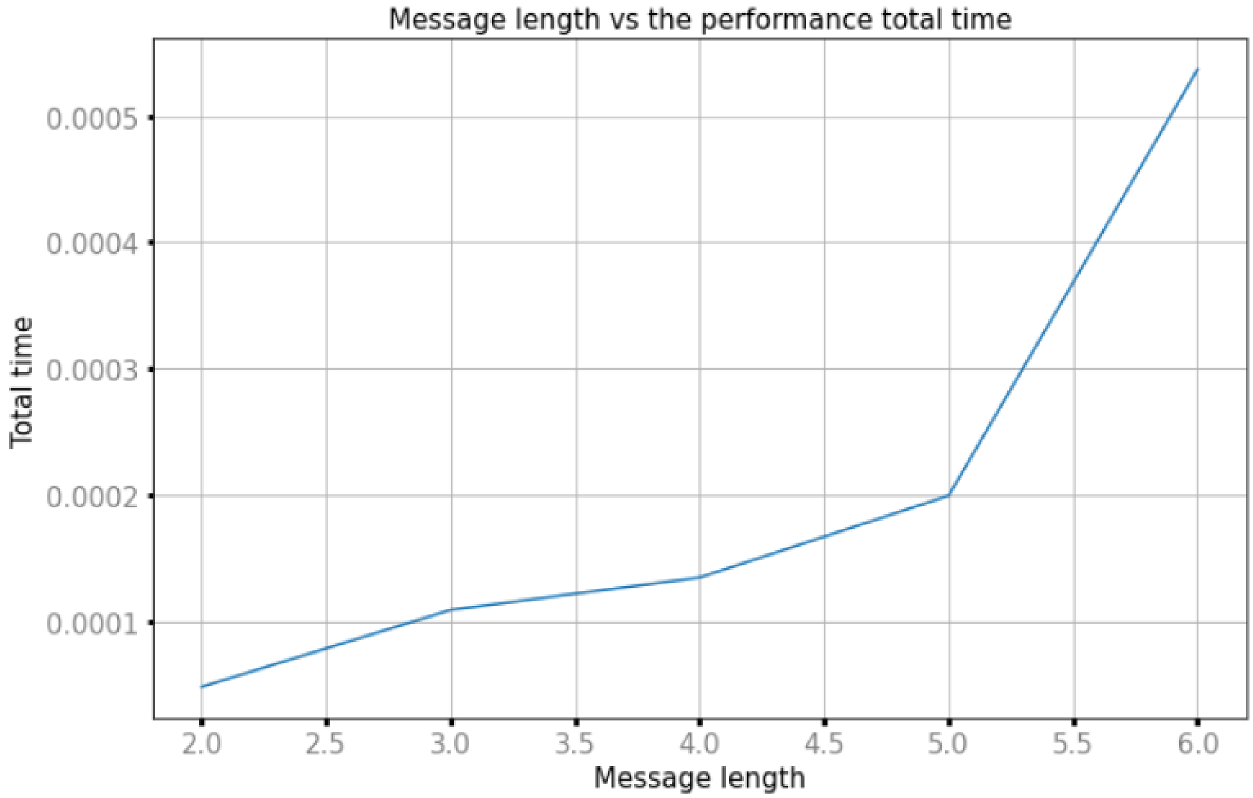
```
$ mpic++ ping_pong_1.cpp -o ping_pong
$ mpirun -np 4 ./ping_pong
```

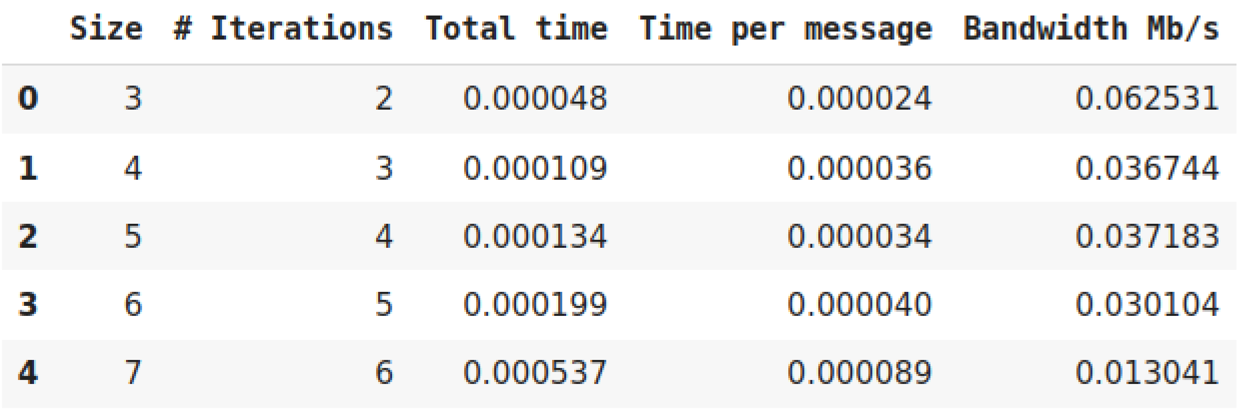
The result is the following. The program prints way of passing the ball and at the end, when game was over, it prints the order of the players.



```
1 0 100 100 100
1 0 100 100 100
PRINT NOT PASSED:
2 3
2
2 0 1 100 100
PRINT NOT PASSED:
3
3
3 0 1 2 100
All players passed the ball!
Game was played in this order:
0
1
2
3
```

Plot and table was made in the file ***ping_pong_2.ipynb.***





	Size	# Iterations	Total time	Time per message	Bandwidth Mb/s
0	3	2	0.000048	0.000024	0.062531
1	4	3	0.000109	0.000036	0.036744
2	5	4	0.000134	0.000034	0.037183
3	6	5	0.000199	0.000040	0.030104
4	7	6	0.000537	0.000089	0.013041

Cellular automata

Boundary conditions

Periodic boundary condition

Values on the boundary are updated as well.

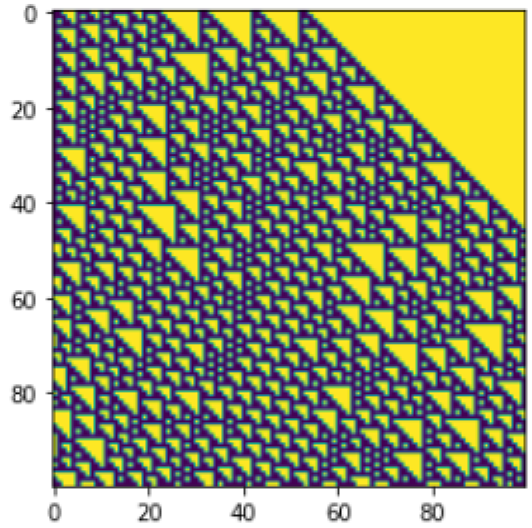
```
void update_cells(vector<int>& cellular_automata)
{
    vector<int> new_cell;
    for (int i = 1; i < cellular_automata.size()-1; i++)
    {
        int sum = cellular_automata[i-1] + cellular_automata[i] + cellular_automata[i+1];
        int sum2 = cellular_automata[i-1] + cellular_automata[i];
        if (sum == 0 || sum == 3 || sum2 == 2)
        {
            new_cell.push_back(1);
        } else
        {
            new_cell.push_back(0);
        }
    }

    cellular_automata.erase(cellular_automata.begin());
    cellular_automata.pop_back();
    cellular_automata.swap(new_cell);
    //new_cell.clear();

}
```

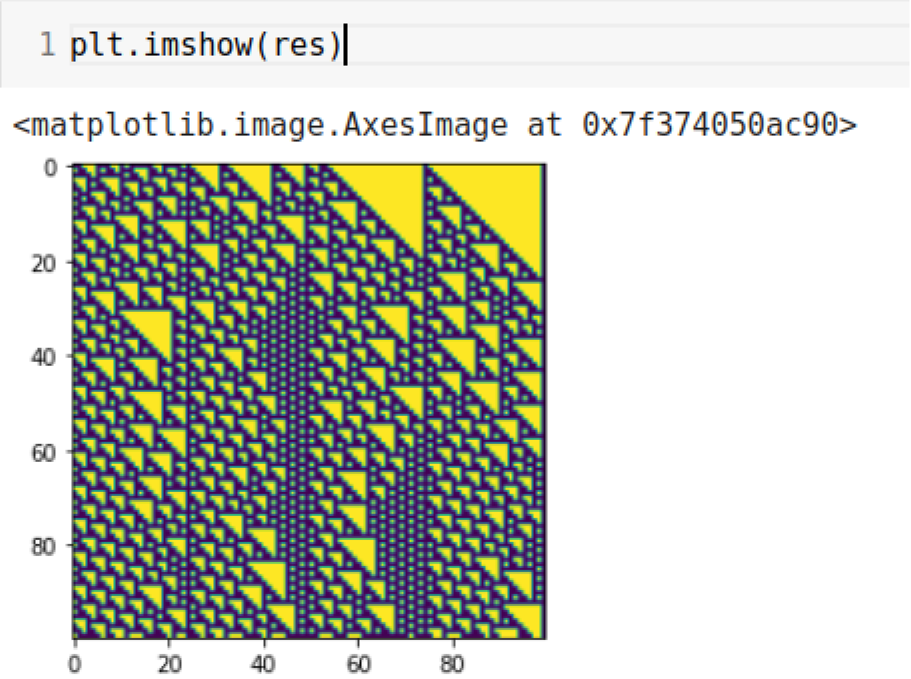
```
plt.imshow(res)
```

```
<matplotlib.image.AxesImage at 0x7f37404f2650>
```



Constant boundary condition

```
void update_cells(vector<int>& cellular_automata)
{
    vector<int> new_cell;
    for (int i = 1; i < cellular_automata.size()-1; i++)
    {
        if (i == 0 || i == cellular_automata.size()-2)
        {
            new_cell.push_back(0);
        } else
        {
            int sum = cellular_automata[i-1] + cellular_automata[i] + cellular_automata[i+1];
            int sum2 = cellular_automata[i-1] + cellular_automata[i];
            if (sum == 0 || sum == 3 || sum2 == 2)
            {
                new_cell.push_back(1);
            } else
            {
                new_cell.push_back(0);
            }
        }
    }
```



```
plt.imshow(res)
```

```
<matplotlib.image.AxesImage at 0x7f374050ac90>
```

Parallelization

```
int main(int argc, char ** argv)
{
    int psize;
    int prank;

    MPI_Status status;

    int ierr;

    int work[1];
    work[0] = 1;

    ierr = MPI_Init(&argc, &argv);
    ierr = MPI_Comm_rank(MPI_COMM_WORLD, &prank);
    ierr = MPI_Comm_size(MPI_COMM_WORLD, &psize);

    int count = 0;
    int N = 20;
    int n_prank = N/psize;
    vector<int> cellular_automata;
    init_vector(cellular_automata, n_prank, prank);


    ...
}
```

Here you can see that chunks initialized in each process. In the end all chunks are gathered in the process 0.

Any kind of rule can be easily input into the computations.

```
void update_cells(vector<int>& cellular_automata)
{
    vector<int> new_cell;
    for (int i = 1; i < cellular_automata.size()-1; i++)
    {
        int sum = cellular_automata[i-1] + cellular_automata[i] + cellular_automata[i+1];
        int sum2 = cellular_automata[i-1] + cellular_automata[i];
        if (sum == 0 || sum == 3 || sum2 == 2)
        {
            new_cell.push_back(1);
        } else
        {
            new_cell.push_back(0);
        }
    }

    cellular_automata.erase(cellular_automata.begin());
    cellular_automata.pop_back();
    cellular_automata.swap(new_cell);
    //new_cell.clear();

}
```

Here there are implemented all 8 rules, you can easily off or on any rule by just erasing or adding in if condition.

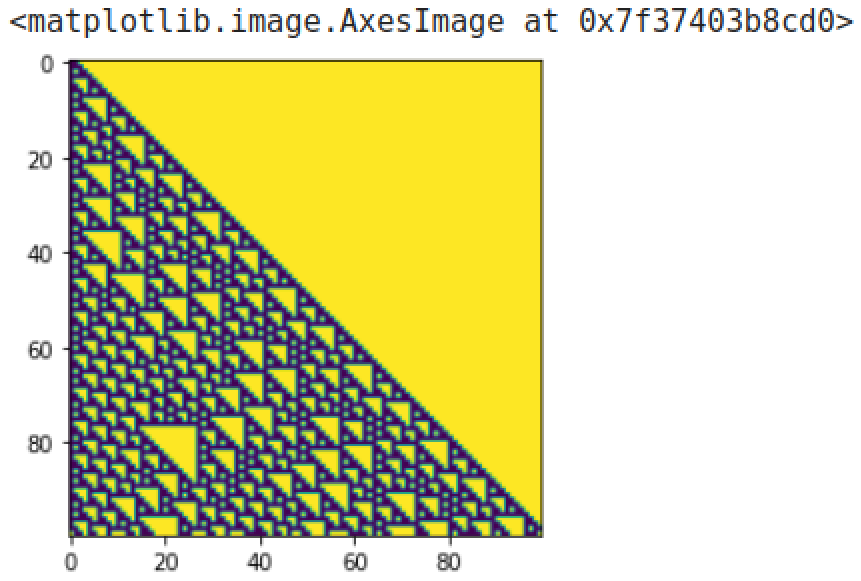
Ghost cells

```
void add_ghost_cells(vector<int>& cellular_automata)
{
    cellular_automata.insert(cellular_automata.cbegin(),100);
    cellular_automata.push_back(100);
}
```

Draw different rules

Using all 8 rules with 3 rules giving 1 – alive.

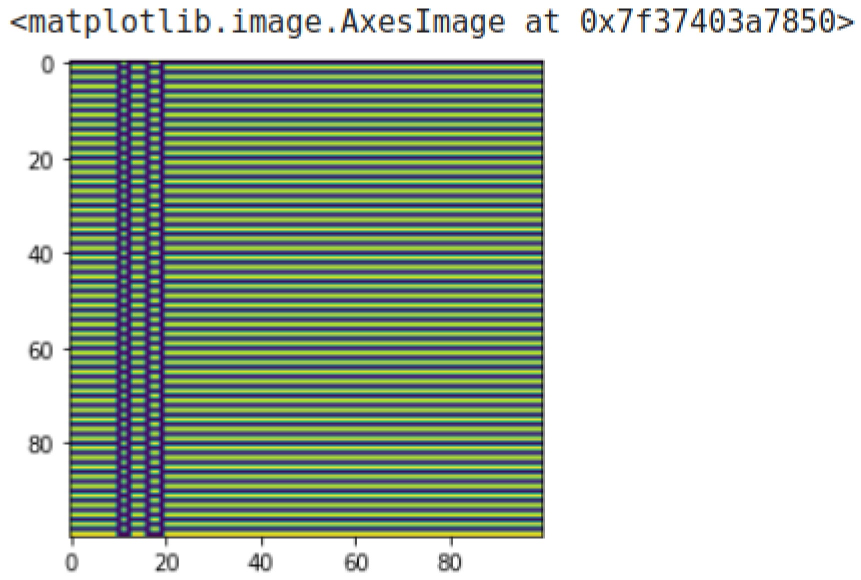
```
plt.imshow(res)
```



```
<matplotlib.image.AxesImage at 0x7f37403b8cd0>
```

Just rule 000 -> 1

```
plt.imshow(res)
```



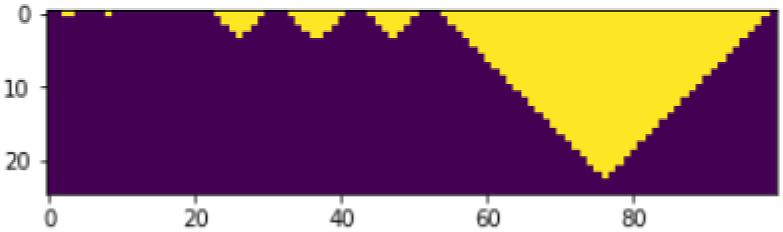
```
<matplotlib.image.AxesImage at 0x7f37403a7850>
```

Just rule 111 -> 1

program stopped, if there is no updates anymore.

```
plt.imshow(res)
```

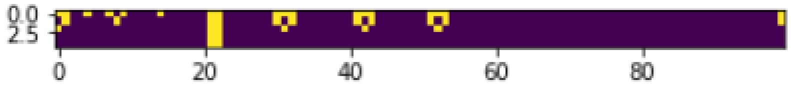
```
<matplotlib.image.AxesImage at 0x7f374030a3d0>
```



Just 110 -> 1

```
plt.imshow(res)
```

```
<matplotlib.image.AxesImage at 0x7f37402e8150>
```


0.0
2.5
0
20
40
60
80