High Performance Python Lab

Term 2 2019/2020

Lecture 5

mpi4py

Outline

- MPI for Python (mpi4py)
- Point-to-point communications
- Collective communications
- Tasks ;)

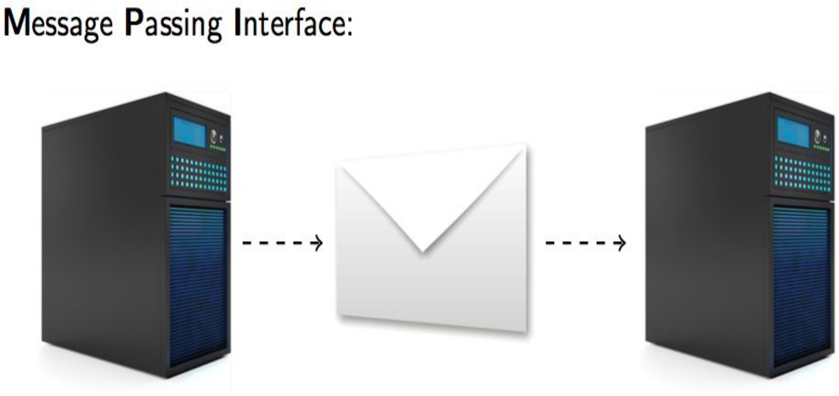
MPI for python (mpi4py)

`$ conda install mpi4py` **or** `$ sudo pip3 install mpi4py`

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

print("Hello world from rank %r!" % rank)
```

```
$ mpirun -n 4 Hellow_world.py
```

Point-to-point communication




```
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

if rank == 0:
    data = { 'a': 7, 'b': 3.14 }
    comm.send(data, dest = 1, tag = 11)

elif rank == 1:
    data = comm.recv(source = 0, tag = 11)
```

Note: Blocking communication!

Point-to-point communication

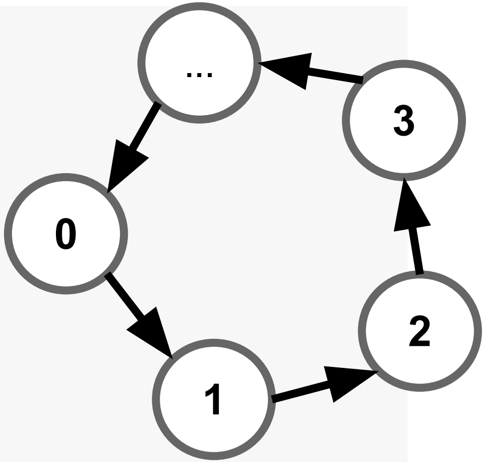
```python
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

# In real code this section might read in data from file
if rank == 0:
    size = 10

    # Send the size of array
    comm.send(size, dest = 1)

    data = np.linspace(0.0, 3.14, size)

    # Send the array itself
    comm.Send(data, dest = 1)

elif rank == 1:
    # Receive the size of array
    size = comm.recv(source = 0)

    # Allocate space to receive the array
    data = np.empty(size, dtype = 'd')

    # Receive the array itself
    comm.Recv(data, source = 0)
```

Point-to-point communication

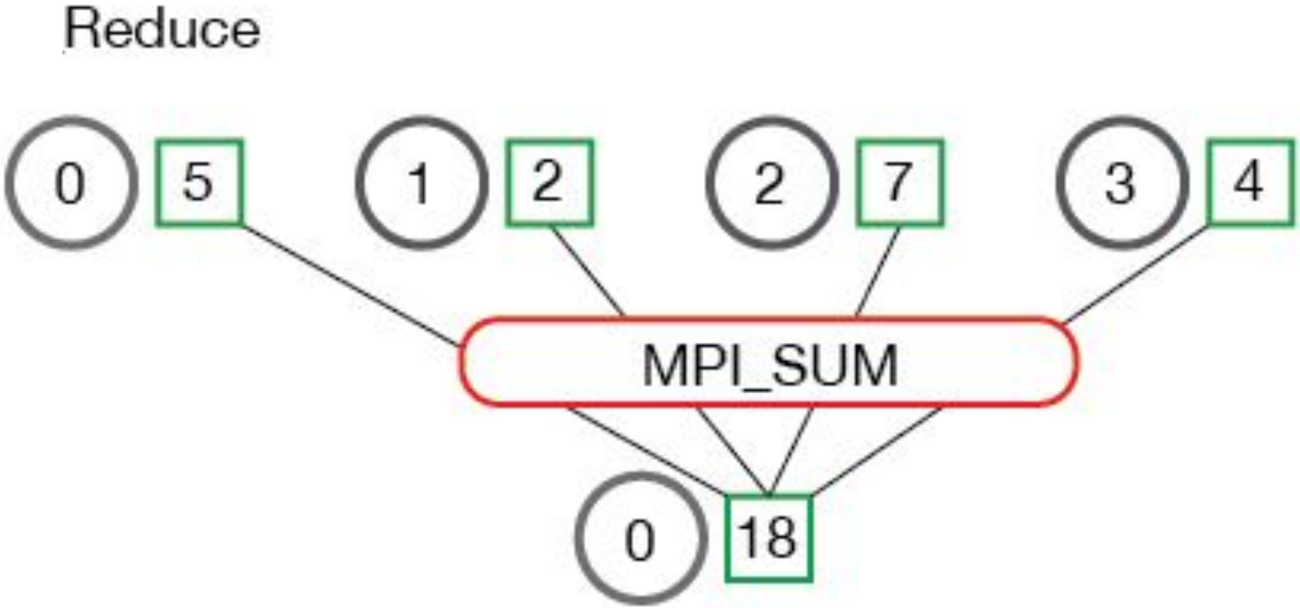
```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

# In real code this section might read in data from file
if rank == 0:
    size = 10

    # Send the size of array
    comm.send(size, dest = 1)

    data = np.linspace(0.0, 3.14, size)

    # Send the array itself
    comm.Send(data, dest = 1)

elif rank == 1:
    # Receive the size of array
    size = comm.recv(source = 0)

    # Allocate space to receive the array
    data = np.empty(size, dtype = 'd')

    # Receive the array itself
    comm.Recv(data, source = 0)
```

Non-blocking point-to-point communication

```python
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

# All processes wait here for all
comm.Barrier()

# Send, not wait
request = comm.isend(rank, dest = (rank + 1) % size)

# Receive, not wait
request = comm.irecv(source = (rank - 1) % size)

# Wait for corresponding sender
data = request.wait()
```



Collective communications: (Sum-)Reduce



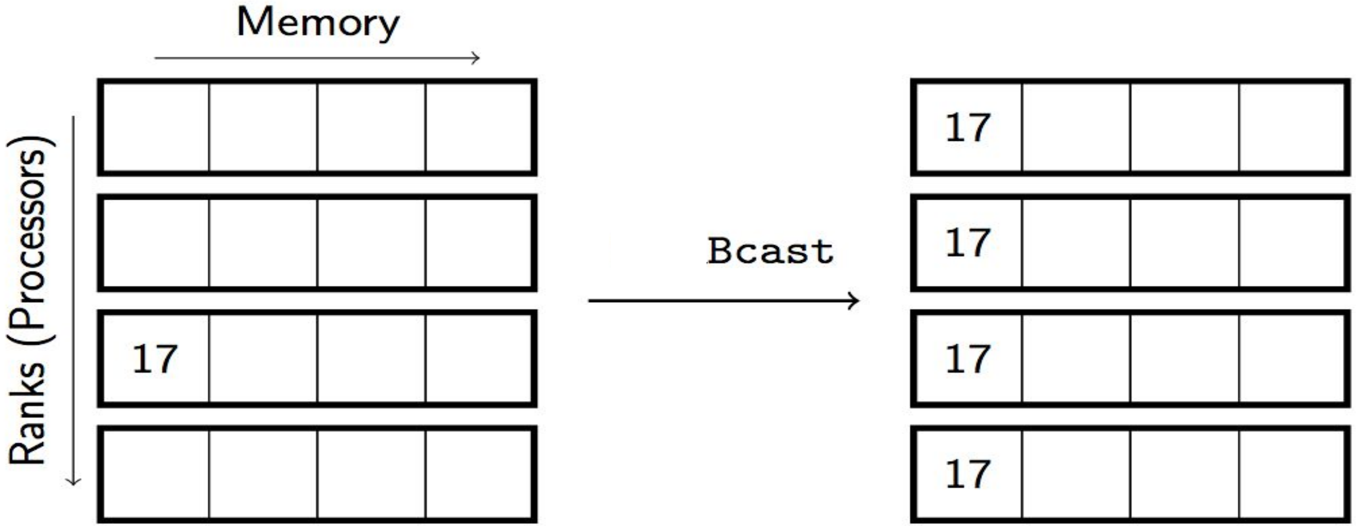
Collective communications: (Sum-)Reduce

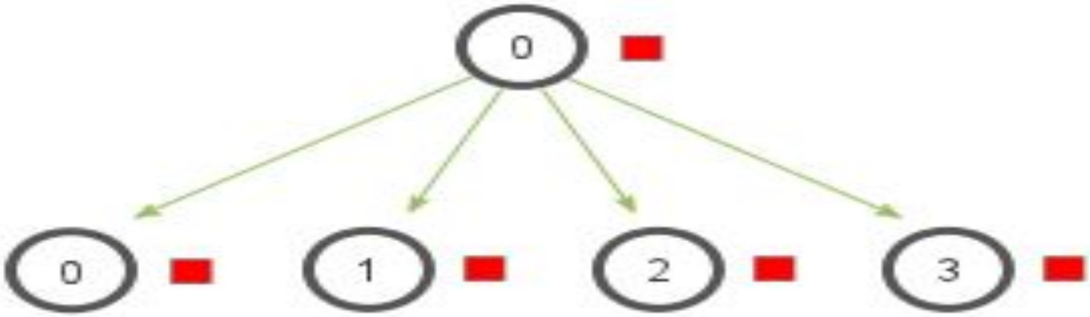
```python
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

# Python objects sum-reduce
result = comm.reduce(rank, op = MPI.SUM, root = 0)


# Numpy objects sum-reduce
sendbuf = np.empty(2, dtype = 'i')
sendbuf[0] = 1
sendbuf[1] = rank

if rank == 0:
    recvbuf = np.empty(2, dtype = 'i')
else:
    recvbuf = None

comm.Reduce(sendbuf, recvbuf, op = MPI.SUM, root = 0)
```

Collective communications: (Sum-)Reduce

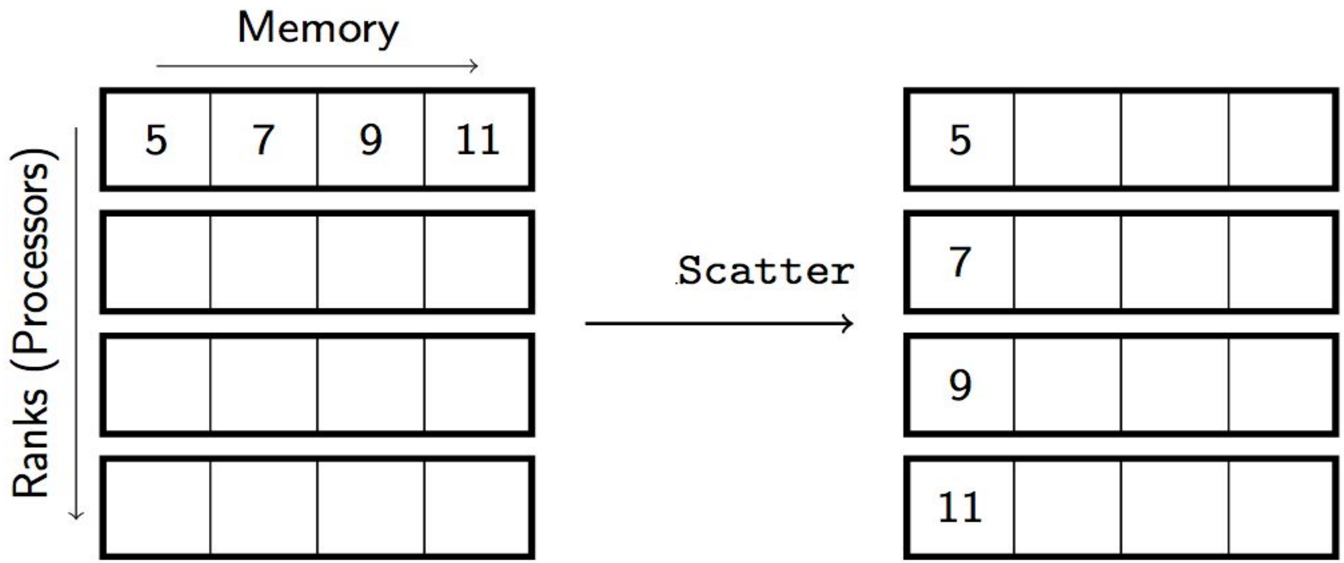
```python
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

# Python objects sum-reduce
result = comm.reduce(rank, op = MPI.SUM, root = 0)


# Numpy objects sum-reduce
sendbuf = np.empty(2, dtype = 'i')
sendbuf[0] = 1
sendbuf[1] = rank

if rank == 0:
    recvbuf = np.empty(2, dtype = 'i')
else:
    recvbuf = None

comm.Reduce(sendbuf, recvbuf, op = MPI.SUM, root = 0)
```

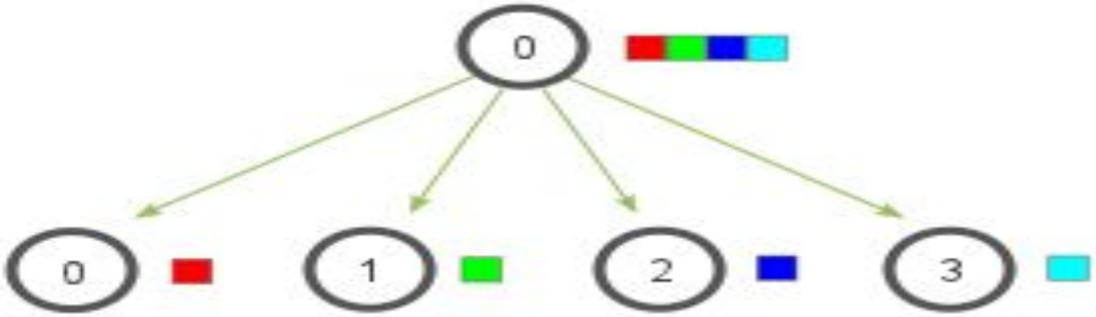
Collective communications: Broadcast



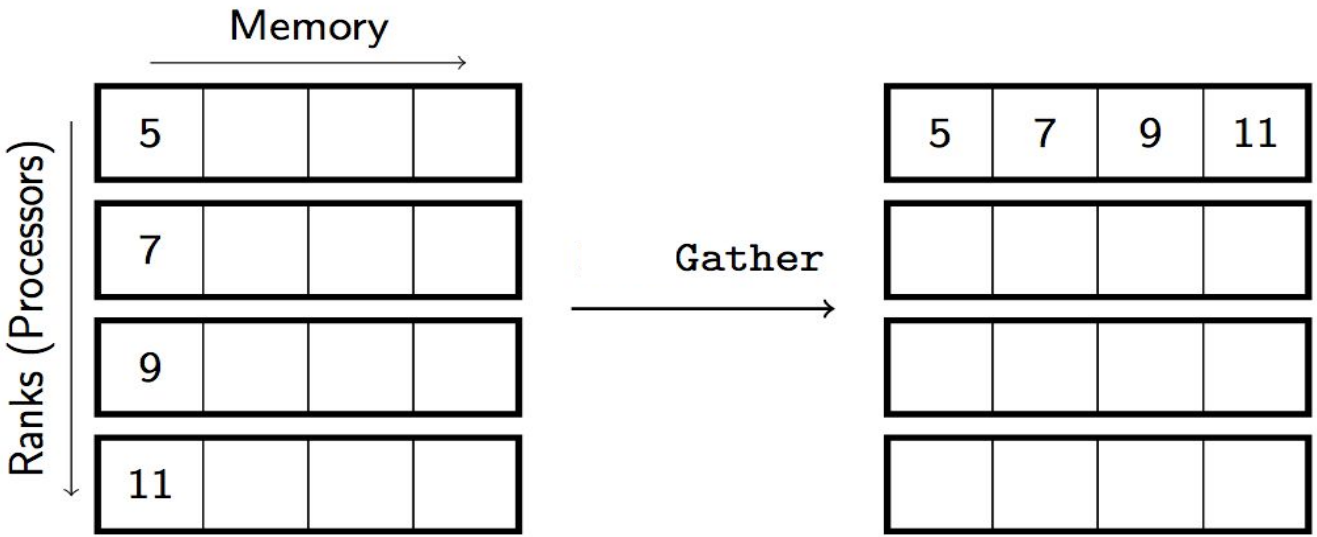


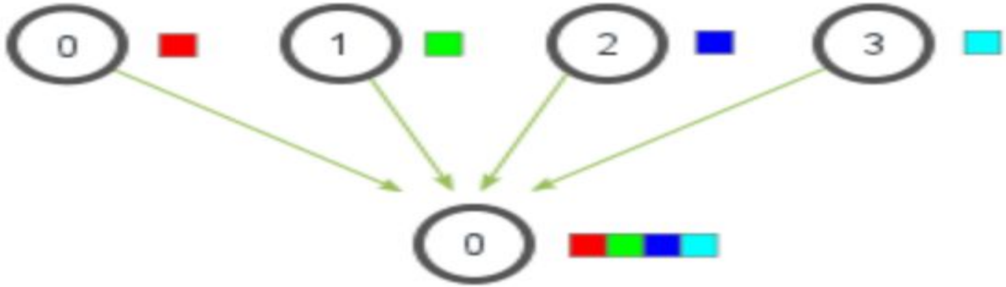
Collective communications: Scatter





Collective communications: Gather



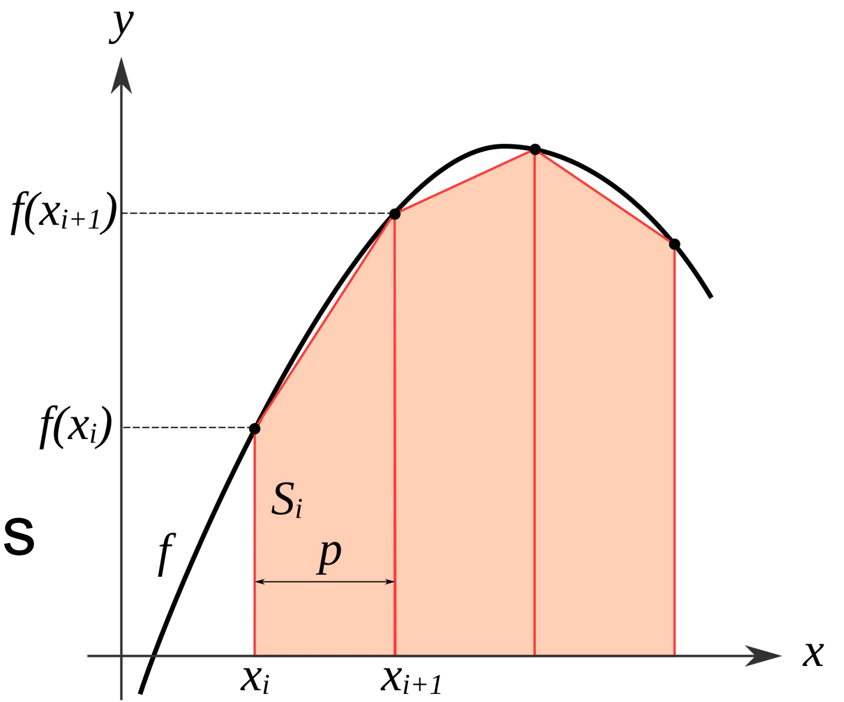


Tasks

- Study an integral
- Columnwise shifted pictures
- Conway’s Game of Life

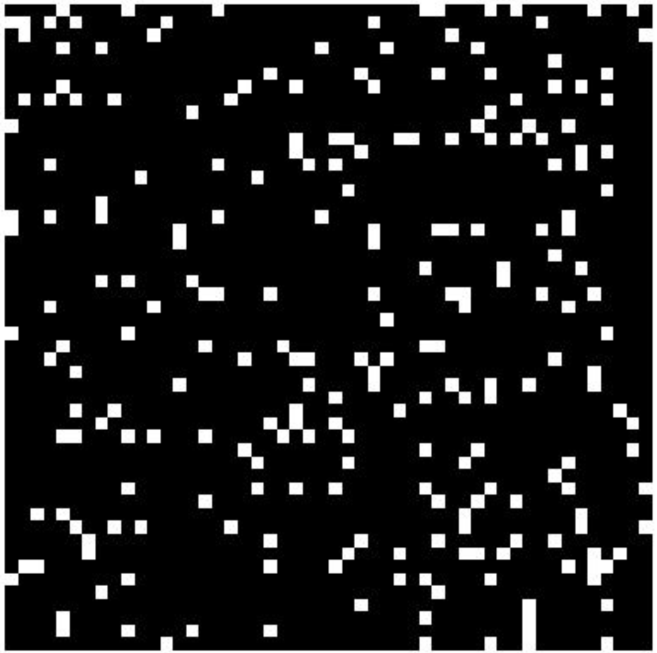
Task 7: Study an integral

- Compute an integral using the trapezoid rule
- Split the job between processes and use **Reduce** collective communication
- Check the accuracy when taking more nodes (i.e. smaller step)
- Check the speedup when taking more processes



Task 8: Columnwise shifteded pictures

- Take a picture
- Split the picture columns between processes
- Shift the columns cyclically

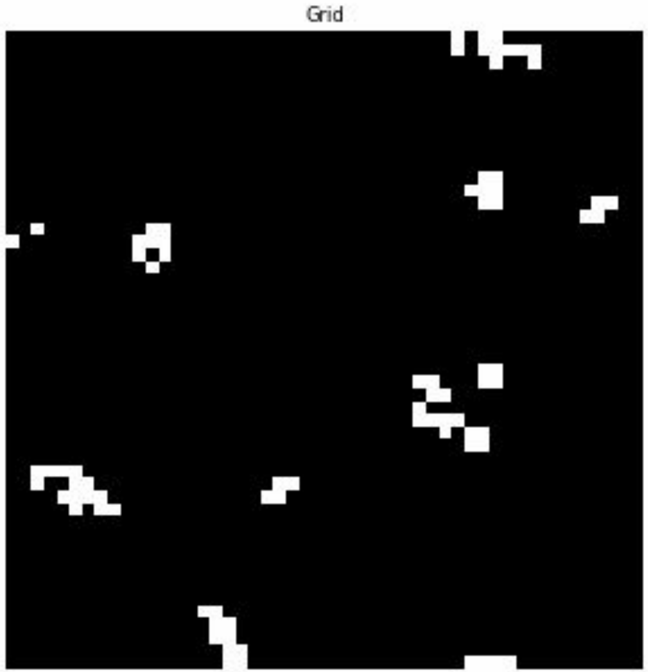


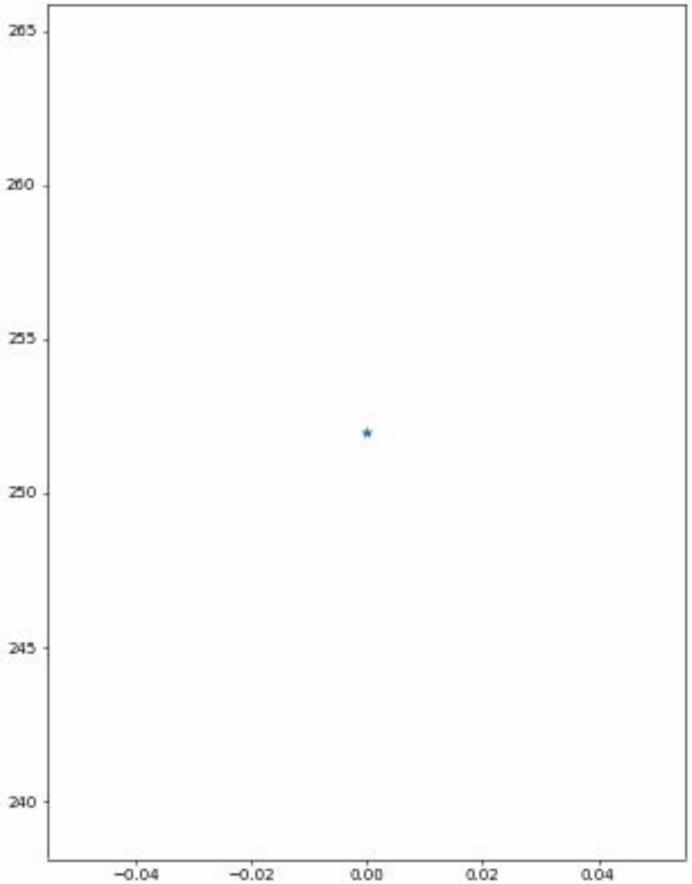
Task 9: Conway's Game of Life

Like Schelling's model, but easier to parallel!

- A grid is a square of N x N cells
- Each cell is either dead or alive
- Each cell has 8 neighbours (the grid is periodic)
- For each cell apply a rule:
    - If cell is dead -- become alive only if exactly 3 neigbours are alive
    - If cell is alive -- stay alive only if it has 2 or 3 alive neighbours

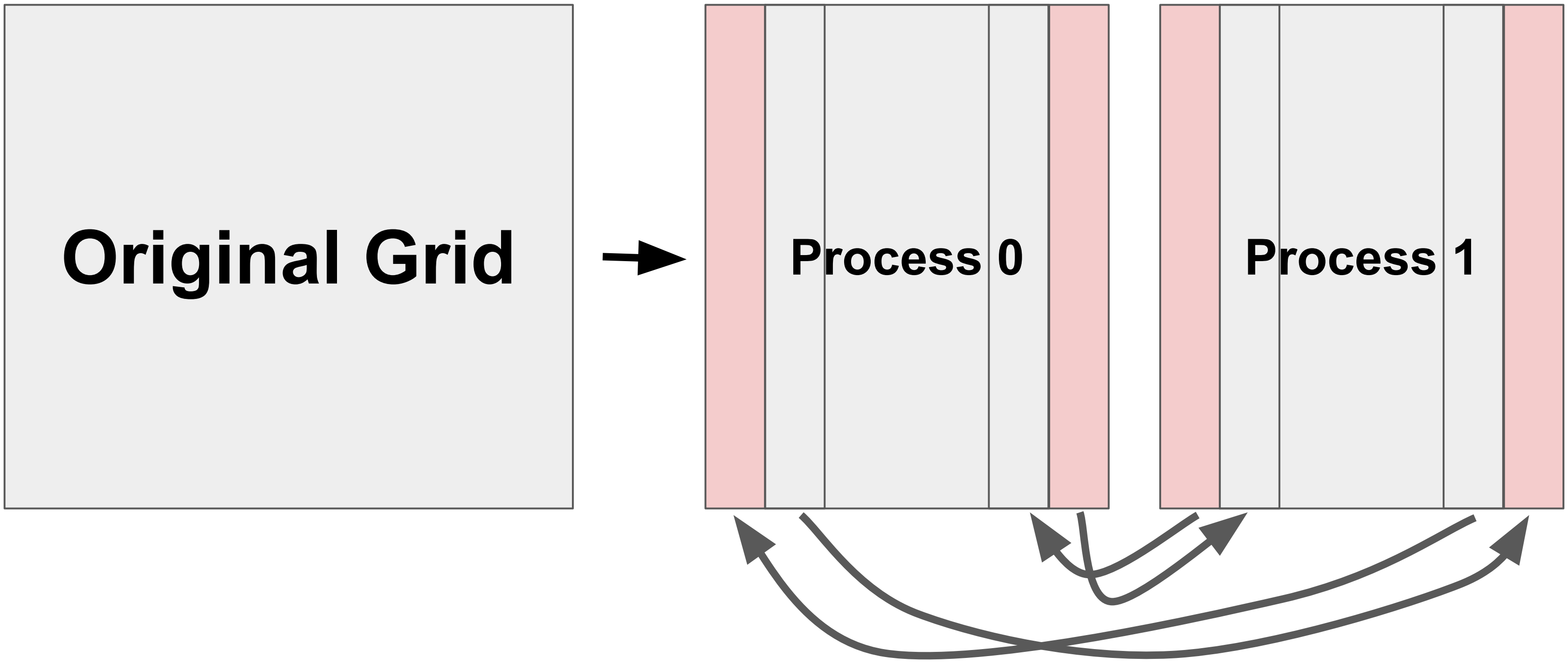
Task 9: Conway’s Game of Life





Task 9: Conway's Game of Life

Parallelize using "ghost" cells (red):

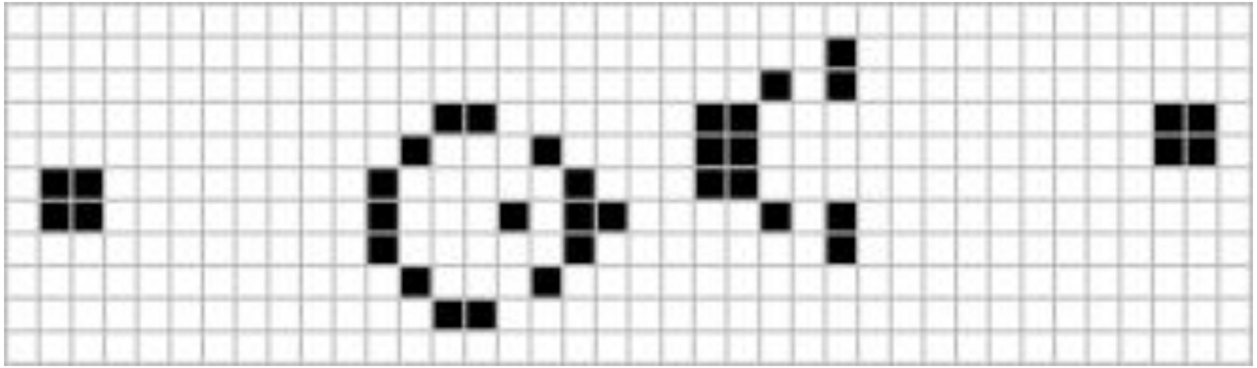


Task 9: Conway’s Game of Life

Try different initial conditions:

For example -- Gosper’s glider gun:

https://tinyurl.com/yx5hy26m



Thank you for your attention!