# Poiseuille flow simulation via Lattice Boltzmann method

Parallel Computing in Mathematical Modeling
and Data-Intensive Applications

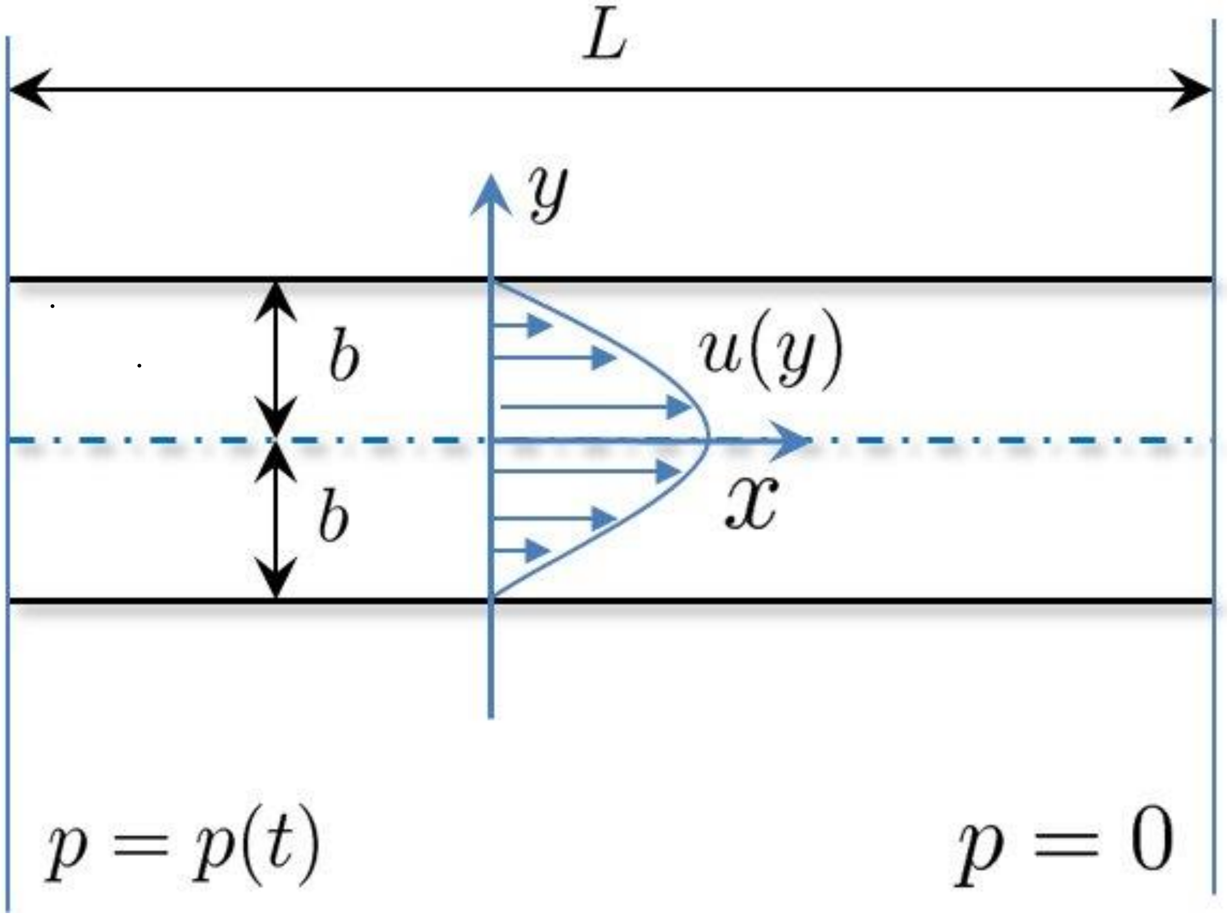
# Task description

- Optimize given sequential Lattice Boltzmann method code
- Develop the parallelized code from sequential
- Profile. Measure timing and plot speedup plot depending on number of processes/threads.
- Conclude. Is the speedup You obtained is approximately linear?

# Task description

Dimensionless parameters of simulation:

| Sign | Description | Value |
|---|---|---|
| $N_x$ | Length of tube, equivalent to L | 1000 |
| $N_y$ | Diameter of tube, equivalent to 2*b | 200 |
| $\rho_{in}$ | Inlet pressure in LBM units. | 1 |
| $\rho_{out}$ | Outlet pressure in LBM units | 0.95 |
| $\tau$ | Relaxation time | 1 |
| $nt$ | Number of time steps | 20000 |



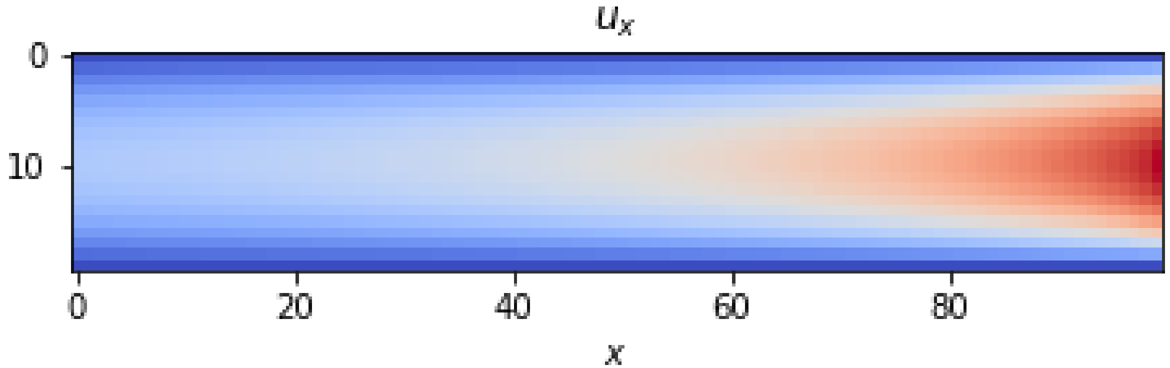
# Task description

$$\widetilde{f_i}(r,t) = f_i(r+\vec{v_i}, t+1)$$ Streaming step

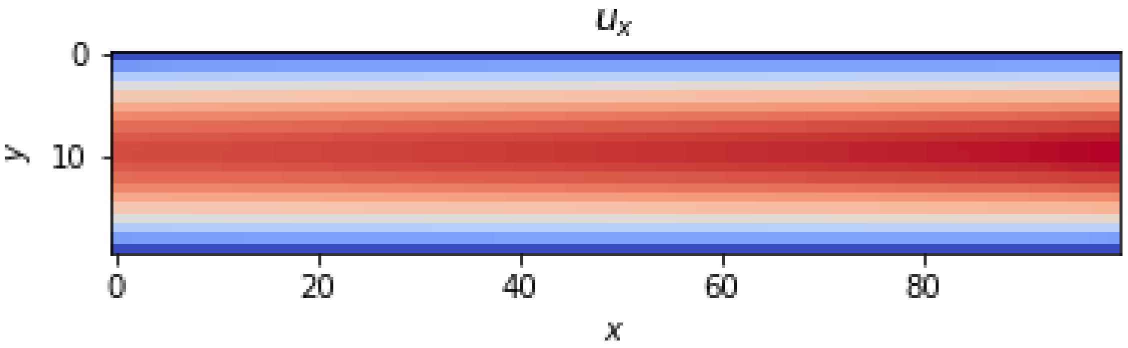
$$f_i(r,t) = \widetilde{f_i}(r,t) - \frac{\widetilde{f_i} - f_i^{eq}}{\tau}$$ Collision step

Poiseuille flow simulation

Unsteady-state




Steady-state




Poiseuille velocity profile:

$$u(y) = -\frac{\Delta p}{4\rho\nu L}(b^2 - y^2)$$

Lattice units:

$$\tau = \frac{\nu}{c_s^2} + 0.5$$

$\nu$ – kinematic viscosity of fluid

$$\rho_{phys} \neq \rho_{LB}$$

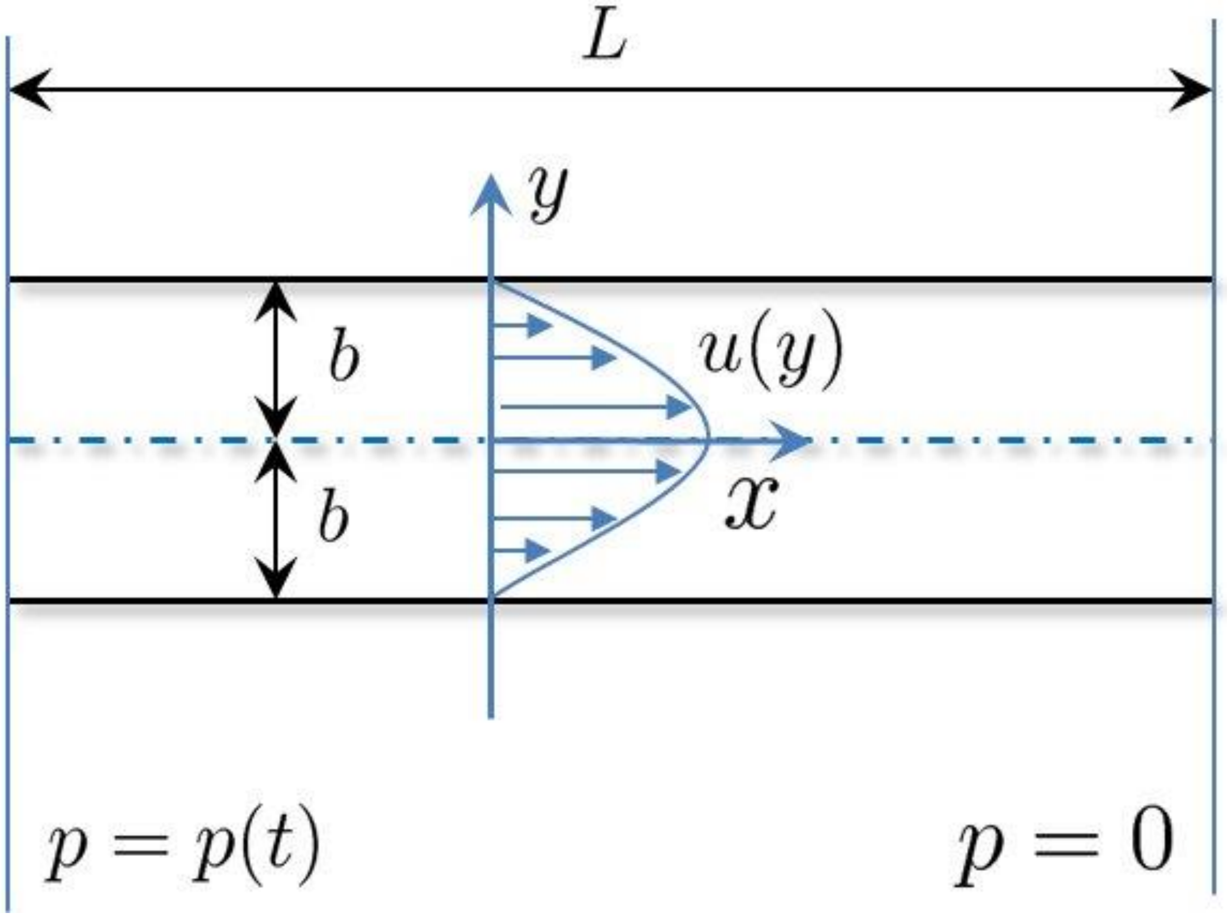
$$p_{LB} = c_s^2 \rho_{LB}$$

Skoltech

# Task description

Several Sequential codes in shared folder:

/gpfs/gpfs0/ParallelComputingShared/Lattice_Boltzmann_2021



# Task description

If (rank < NX % nprocs): //ranks that are less than reminder

rank_nx = NX/nprocs+1;

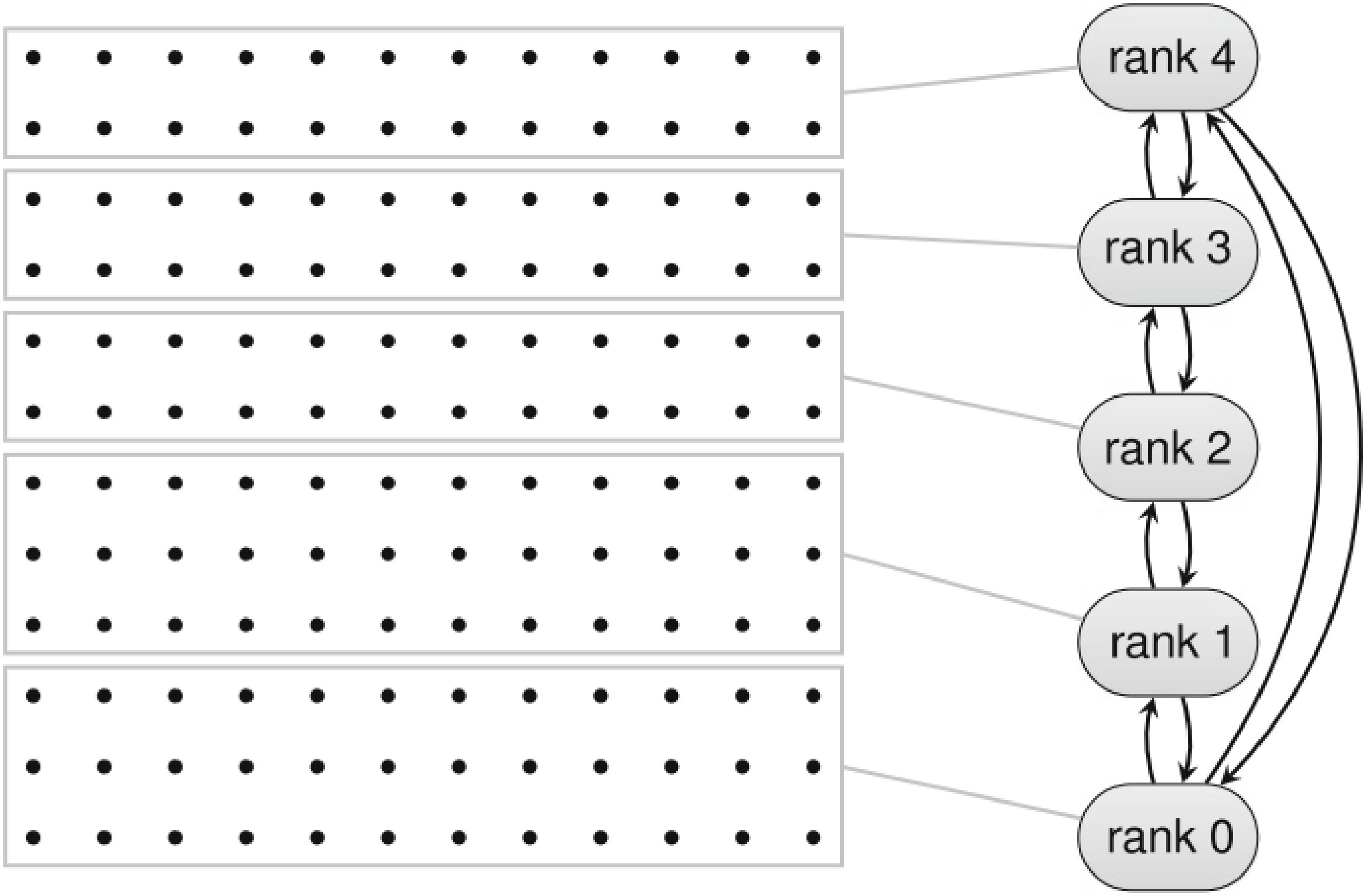
rank_xstart = rank*rank*nx;

else:

rank_nx = NX/procs;

rank_xstart = NX-(nprocs-rank)*rank_nx

print("Rank %d: %d nodes from y = %d to y = %d\n", % (rank, rank_nx, rank_xstart, rank_xstart+rank_nx-1))



# Task description

```python
# streaming and collision step
def streaming_and_collision(f):
    # streaming
    for j in range(Ny-1, 0, -1):
        for i in range(0, Nx-1):
            f[i,j,2] = f[i,j - 1,2]
            f[i,j,6] = f[i + 1,j - 1,6]
```

```
def streaming_and_collision(f, ranknx, rank_xstart):
        …
        for j in range(Ny-1,0,-1):
                for I in range(rank_xstart, ranknx):
```

# Blocking realization

```
if(rank % 2 == 0) // even ranks send then receive
{
  // send below, i.e. rank-1
  MPI_Send(send_buffer, count, MPI_DOUBLE,
           rankm1, tag, MPI_COMM_WORLD);
  // receive from above, i.e. rank+1
  MPI_Recv(recv_buffer, count, MPI_DOUBLE,
           rankp1, tag, MPI_COMM_WORLD,
           status);
}
else // odd ranks receive then send

{
  // receive from above, i.e. rank+1
  MPI_Recv(recv_buffer, count, MPI_DOUBLE,
           rankp1, tag, MPI_COMM_WORLD,
           status);
  // send below, i.e. rank-1
  MPI_Send(send_buffer, count, MPI_DOUBLE,
           rankm1, tag, MPI_COMM_WORLD);
}
```

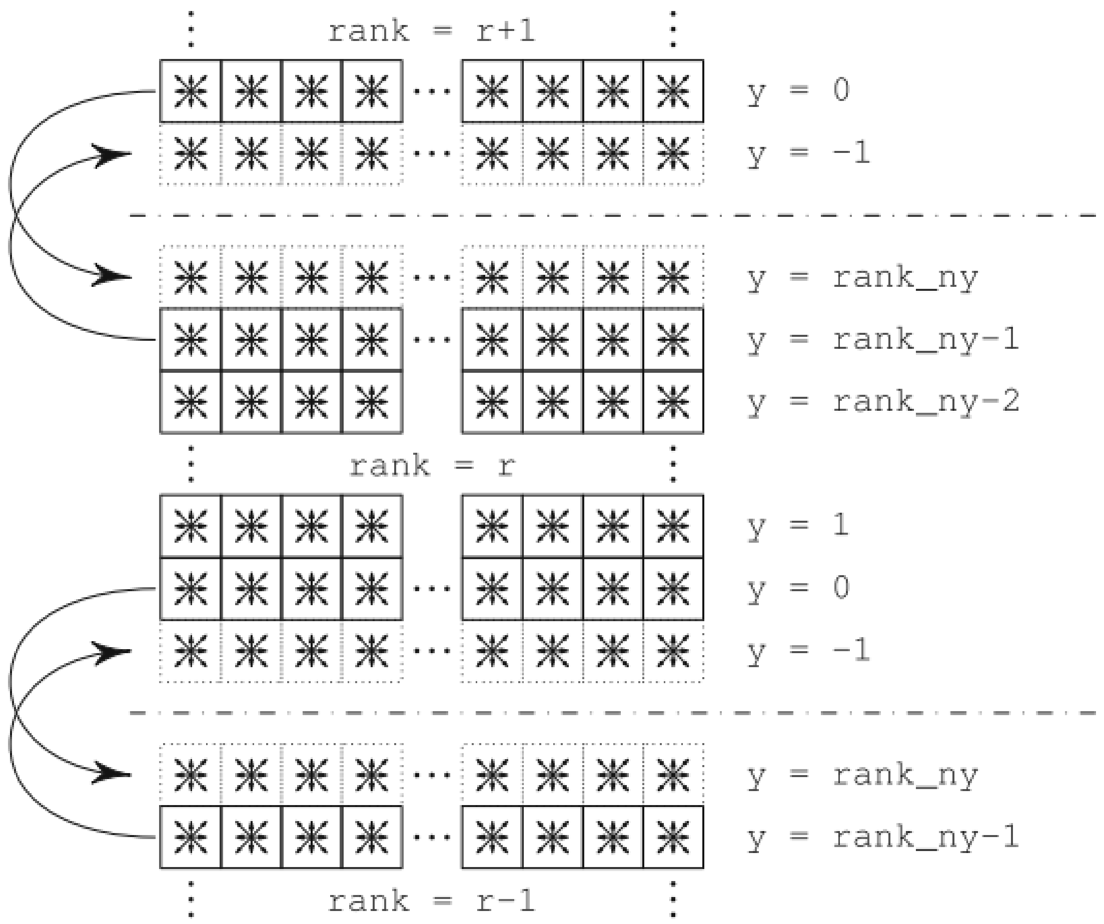



**Fig. 13.10** Diagram showing the rows of data that need to be transferred across subdomain boundaries between processes. *Boxes with solid outlines* denote the nodes of the subdomain updated by each rank, while *boxes with dotted outlines* denote the extra rows used to store data from adjacent subdomains that are handled by different ranks

# Blocking realization

```
MPI_Sendrecv(&f1[fieldn_index(0,rank_ny-1,1)],
             transfer_doubles,MPI_DOUBLE,
             rankp1,rank,
             &f1[fieldn_index(0,-1,1)],
             transfer_doubles,MPI_DOUBLE,
             rankm1,rankm1,
             MPI_COMM_WORLD,MPI_STATUS_IGNORE);

MPI_Sendrecv(&f1[fieldn_index(0, 0,1)],
             transfer_doubles,MPI_DOUBLE,
             rankm1,rank,
             &f1[fieldn_index(0,rank_ny,1)],
             transfer_doubles,MPI_DOUBLE,
             rankp1,rankp1,
             MPI_COMM_WORLD,MPI_STATUS_IGNORE);
```

```
size_t transfer_doubles = (ndir-1)*NX;
```

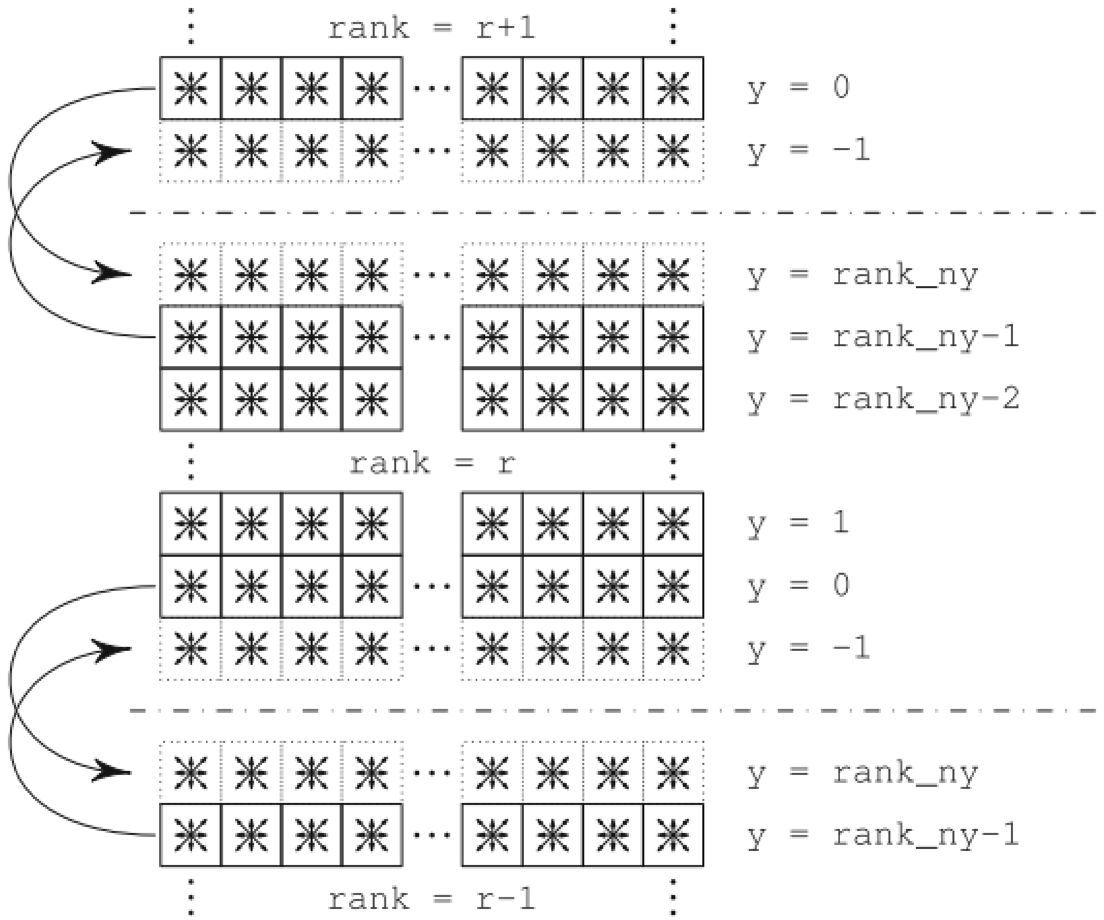



**Fig. 13.10** Diagram showing the rows of data that need to be transferred across subdomain boundaries between processes. *Boxes with solid outlines* denote the nodes of the subdomain updated by each rank, while *boxes with dotted outlines* denote the extra rows used to store data from adjacent subdomains that are handled by different ranks

Poiseuille flow simulation

Skoltech

# Non-Blocking realization

```
// Start a nonblocking send
int MPI_Isend(const void *buf,
              int count, MPI_Datatype datatype,
              int dest, int tag,
              MPI_Comm comm, MPI_Request *request

// Start a nonblocking receive
int MPI_Irecv(void *buf,
              int count, MPI_Datatype datatype,
              int source, int tag,
              MPI_Comm comm, MPI_Request *request
```

```
MPI_Request  reqs[4];
MPI_Status   stats[4];
```

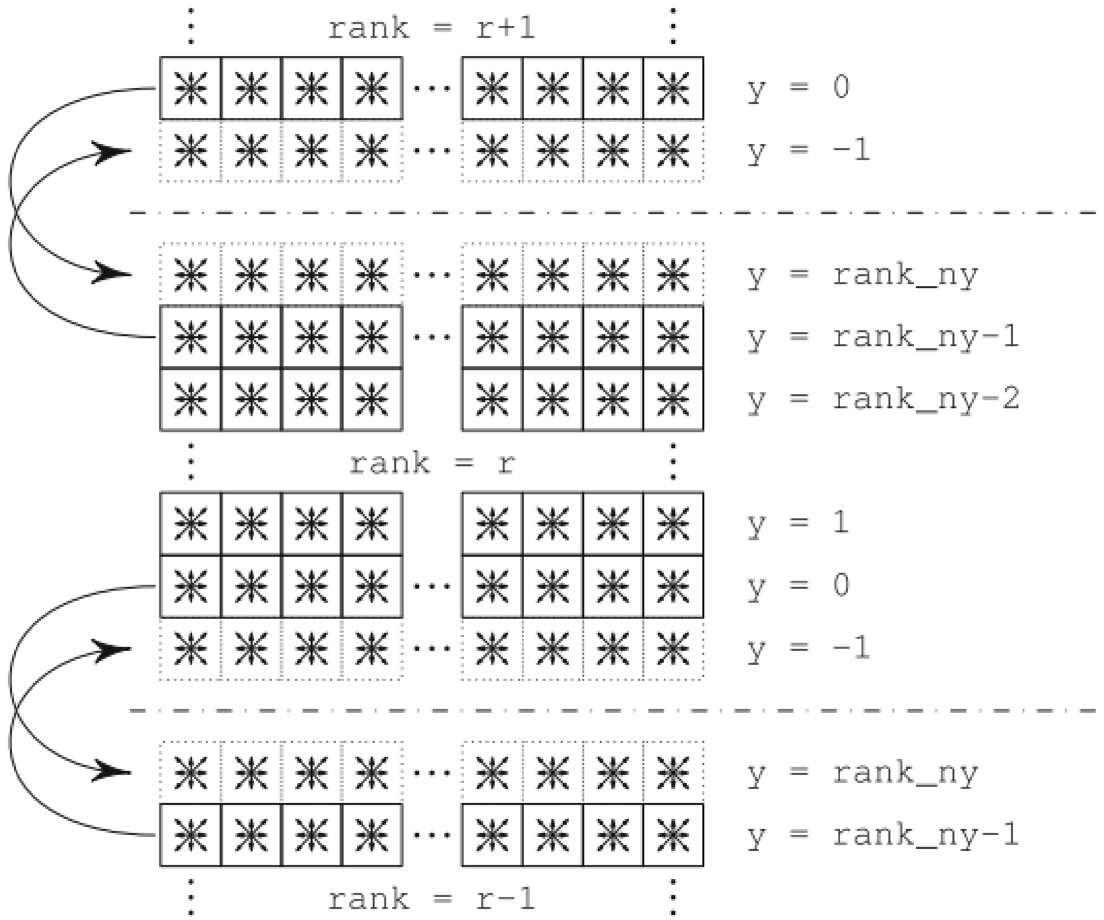



**Fig. 13.10** Diagram showing the rows of data that need to be transferred across subdomain boundaries between processes. *Boxes with solid outlines* denote the nodes of the subdomain updated by each rank, while *boxes with dotted outlines* denote the extra rows used to store data from adjacent subdomains that are handled by different ranks

Poiseuille flow simulation

Skoltech

# Non-Blocking realization

```
MPI_Request  reqs[4];
MPI_Status   stats[4];
```

```
MPI_Isend(&f1[fieldn_index(0,rank_ny-1,1)],
          transfer_doubles,MPI_DOUBLE,
          rankp1,rank,  MPI_COMM_WORLD,&reqs[0]
MPI_Irecv(&f1[fieldn_index(0,-1,1)],
          transfer_doubles,MPI_DOUBLE,
          rankm1,rankm1,
          MPI_COMM_WORLD,&reqs[1]);

MPI_Isend(&f1[fieldn_index(0,0,1)],
          transfer_doubles,MPI_DOUBLE,
          rankm1,rank,
          MPI_COMM_WORLD,&reqs[2]);
MPI_Irecv(&f1[fieldn_index(0,rank_ny,1)],
          transfer_doubles,MPI_DOUBLE,
          rankp1,rankp1,
          MPI_COMM_WORLD,&reqs[3]);
```

```
int MPI_Waitall(int count,
                MPI_Request *array_of_requests,
                MPI_Status *array_of_statuses)
```

```
int MPI_Testall(int count,
                MPI_Request *array of requests,
                int *flag,
                MPI_Status *array of statuses)
```

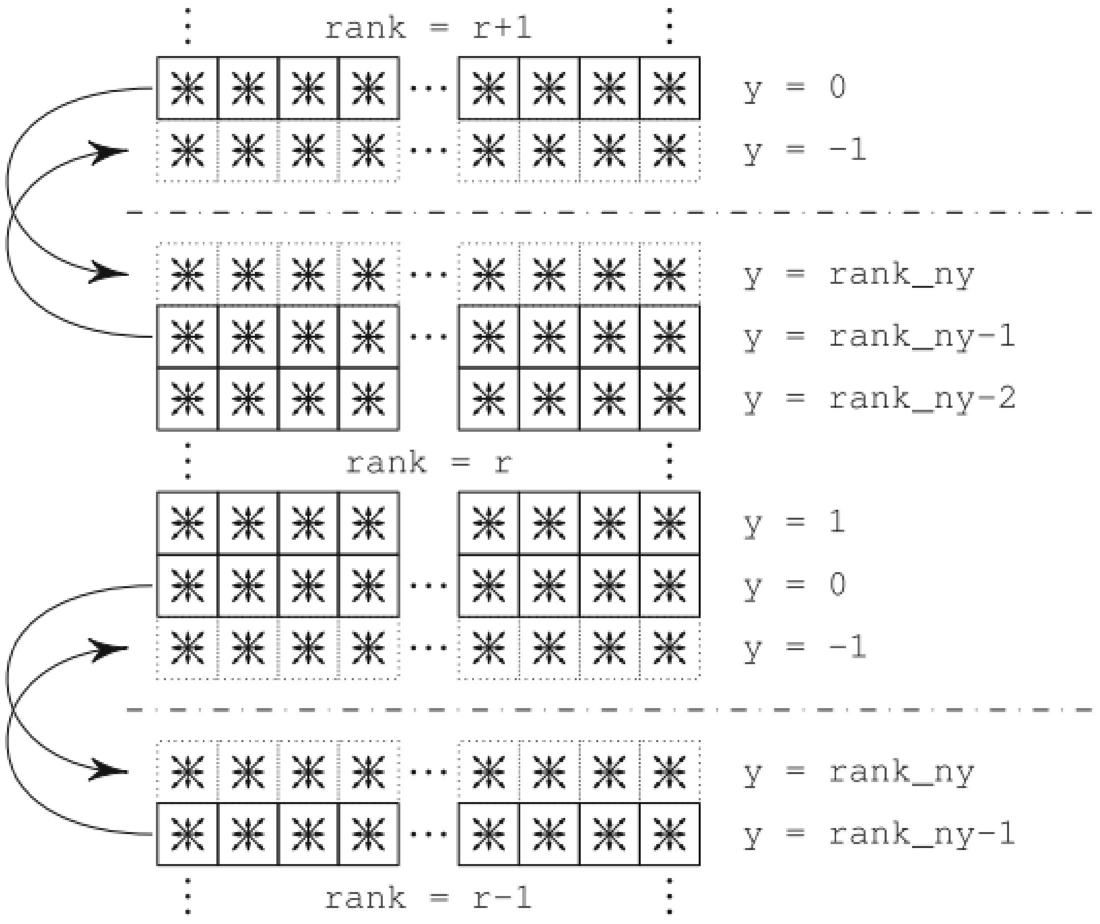



**Fig. 13.10** Diagram showing the rows of data that need to be transferred across subdomain boundaries between processes. *Boxes with solid outlines* denote the nodes of the subdomain updated by each rank, while *boxes with dotted outlines* denote the extra rows used to store data from adjacent subdomains that are handled by different ranks

Poiseuille flow simulation

Skoltech

# Non-Blocking realization

## Profiling tools for Python

CPU time profiling tools:
1. timeit
2. cProfile
3. line_profiler

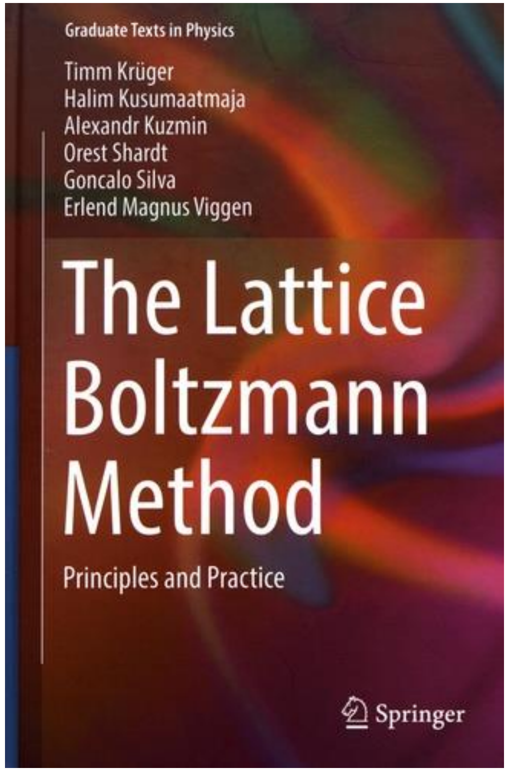
Memory profiling tools:
1. memory_profiler
2. heapy

*Contacts for questions:*
*Telegram: @AndreyOlhin*
*Email: Andrey.Olhin@skoltech.ru*

## MPI commands for Python:

/gpfs/gpfs0/ParallelComputingShared/Lattice_Boltzmann_2021 MPI_commands/

## Lattice Boltzmann book




*Principles of parallelization of LBM – Chapter 13.4*



Skoltech

# thx.