<1> System. advance( )

System: Construct variable

System. compute_diagnostics( )

System. compute_histogram( )

D : diffusion constant

L : domain length

N : grid points per direction

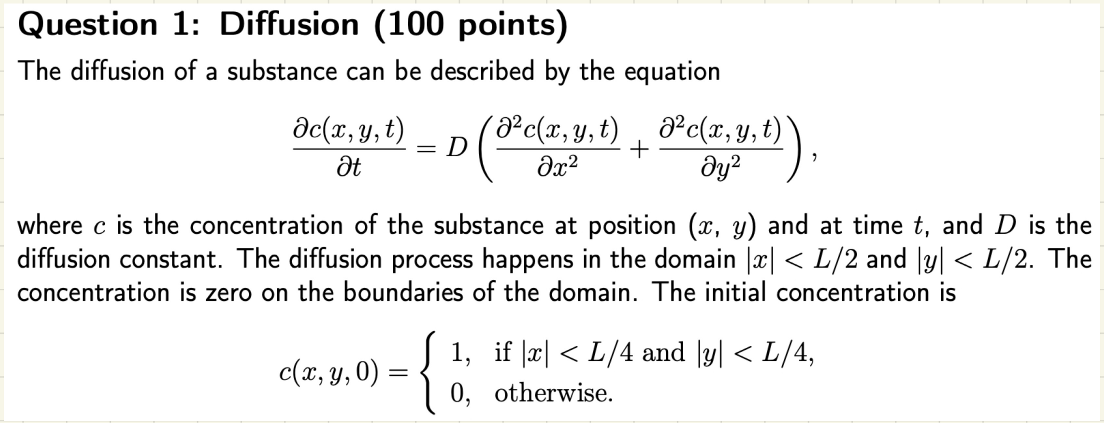
local N : number of rows of this process

h : grid spacing

dt : timestep

aux : auxiliary variable

c : solution vector



Question 1: Diffusion (100 points)

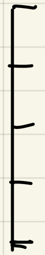
The diffusion of a substance can be described by the equation

$$\frac{\partial c(x,y,t)}{\partial t} = D\left(\frac{\partial^2 c(x,y,t)}{\partial x^2} + \frac{\partial^2 c(x,y,t)}{\partial y^2}\right),$$

where c is the concentration of the substance at position $(x,\ y)$ and at time t, and D is the diffusion constant. The diffusion process happens in the domain $|x| < L/2$ and $|y| < L/2$. The concentration is zero on the boundaries of the domain. The initial concentration is

$$c(x,y,0) = \begin{cases} 1, & \text{if } |x| < L/4 \text{ and } |y| < L/4, \\ 0, & \text{otherwise.} \end{cases}$$

local_N = N / size; ⇒ 不整除?



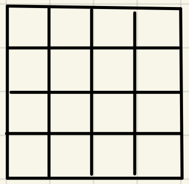
```
void advance( )
{

}
```

$\Downarrow$

在时间步上更新一步，结果存在Ctemp里。

```
void compute_diagnostics( t )
```

计算某一时刻 total concentration



$c \cdot d^2$ 求和

```
void compute_histogram( )
```

找到浓度最大和最小的点，并等距分段，从而画出 histgram.

void initialize( )

Initialization of the pde

a) The skeleton code solves the equation on a uniform grid using a central finite difference scheme in space and forward Euler time integration. Parallellize the code by filling parts marked by `TODO` in the functions `advance` and `main`. Use a tiling decomposition scheme (i.e., distribute the rows evenly to the MPI processes). How to run the code:

- `make` to compile the code
- `make run` to run single core (please not in the login node on euler!). If not locally on the laptop, use `sbatch launch_single.sh` to submit a job via slurm.
- to run multicore use: `mpirun -n x ./diffusion D L N`. You can also use `sbatch launch_single.sh` to submit a job via slurm.

Solution:

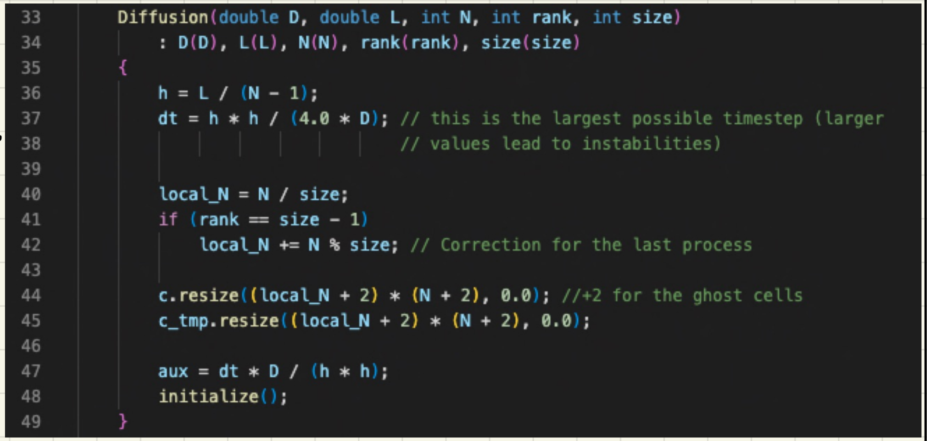
- main part:

195 ; 247]

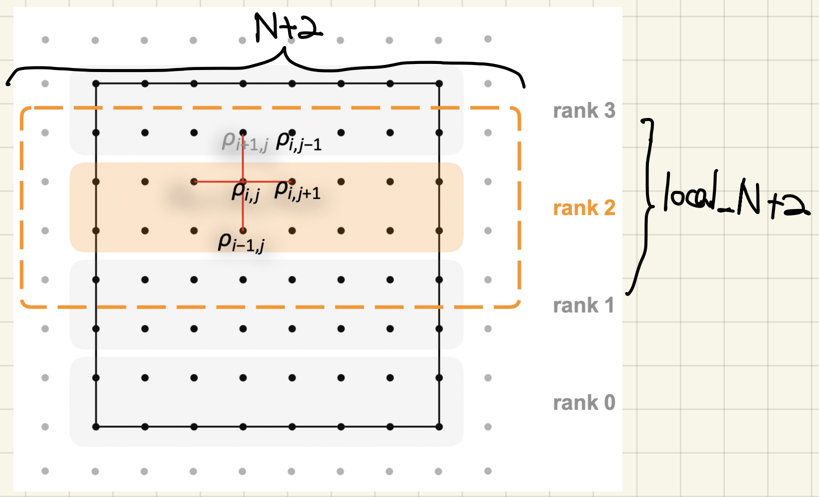
- advance part:

Analyzing

- C is used for storing information.
- local_N



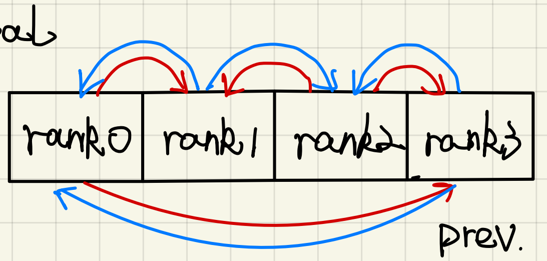
```
Diffusion(double D, double L, int N, int rank, int size)
    : D(D), L(L), N(N), rank(rank), size(size)
{
    h = L / (N - 1);
    dt = h * h / (4.0 * D); // this is the largest possible timestep (larger
                            // values lead to instabilities)

    local_N = N / size;
    if (rank == size - 1)
        local_N += N % size; // Correction for the last process

    c.resize((local_N + 2) * (N + 2), 0.0); //+2 for the ghost cells
    c_tmp.resize((local_N + 2) * (N + 2), 0.0);

    aux = dt * D / (h * h);
    initialize();
}
```



C: [ ] → one-dimension vector

↓

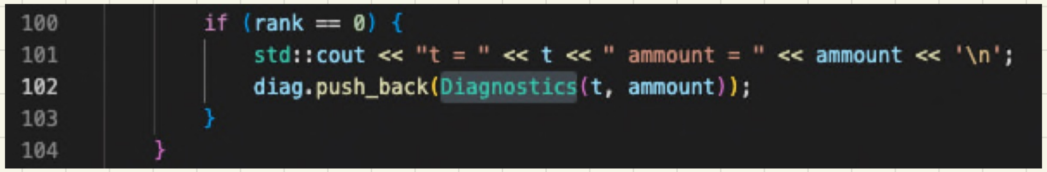
goal



for a specific rank:

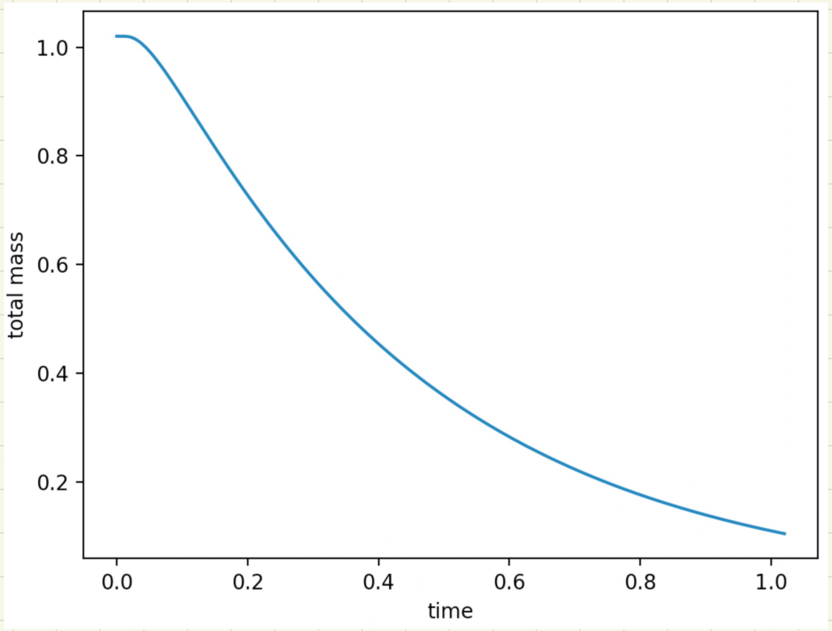
1. Don't know how to enforce Dirichlet B.C.
2. Don't know the use of MPI_PROC_NULL

b) For a given time compute the integral of $c(x, y, t)$ over the domain (total ammount of the substance). Fill the missing MPI parts in `compute_diagnostics`, and plot the result as a function of time using $D = 1$, $L = 2$ and $N = 100$. When run correctly, the code will output file called `diagnostics.dat`. To plot this data use `python plot_diagnostics.py` (module load python).



```
        if (rank == 0) {
            std::cout << "t = " << t << " amount = " << amount << '\n';
            diag.push_back(Diagnostics(t, amount));
        }
    }
```

• change this part: ammount → total_sum



c) For a given time compute the histogram of $c(x, y, t)$ in the function `compute_histogram` by implementing the missing MPI parts marked by `TODO`, and plot or print the resulting histogram for $t = 0.5$ using $D = 1$, $L = 2$ and $N = 100$.

• Don't know how to exchange the date.
The code seems to have some problems.

d) Suggest other ways to divide the real-space domain between processes with the aim of minimizing communication overhead. Prove your argument by computing the message communication size for the tiling domain decomposition and for your suggestion.

1. What's the meaning of communication overhead.
通信开销

! no idea

e) Make a strong and weak scaling plot up to 48 cores. Justify what is happening in your plots. Make at least five different numbers of cores runs (e.g. [1, 12, 24, 36, 48] or [1, 2, 4, 8, 16]). For the strong scaling plot use: N = {1024, 2048, 4096, 8192}, in other words, plot at least four lines (if too slow use smaller N's). For the weak scaling plot, use N = 1024 and N = 2048 (if those are taking too long use smaller N's). Use D = 1, L = 2 and modify the number of timesteps `step = 100`. Do not forget to state which CPU you ran the tests on! You can use the `run.sh` script to run for different number of cores. On euler use `run.sh` via `sbatch launch.sh`. Draw the plots either directly on the paper (which is the way you will do it on the exam). Or use any desired ploting scripts.

· review the weak/strong scaling.