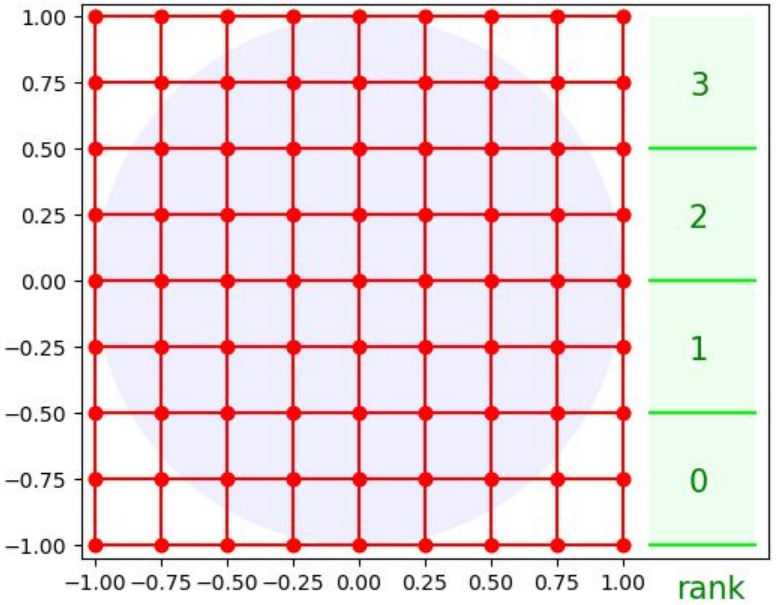
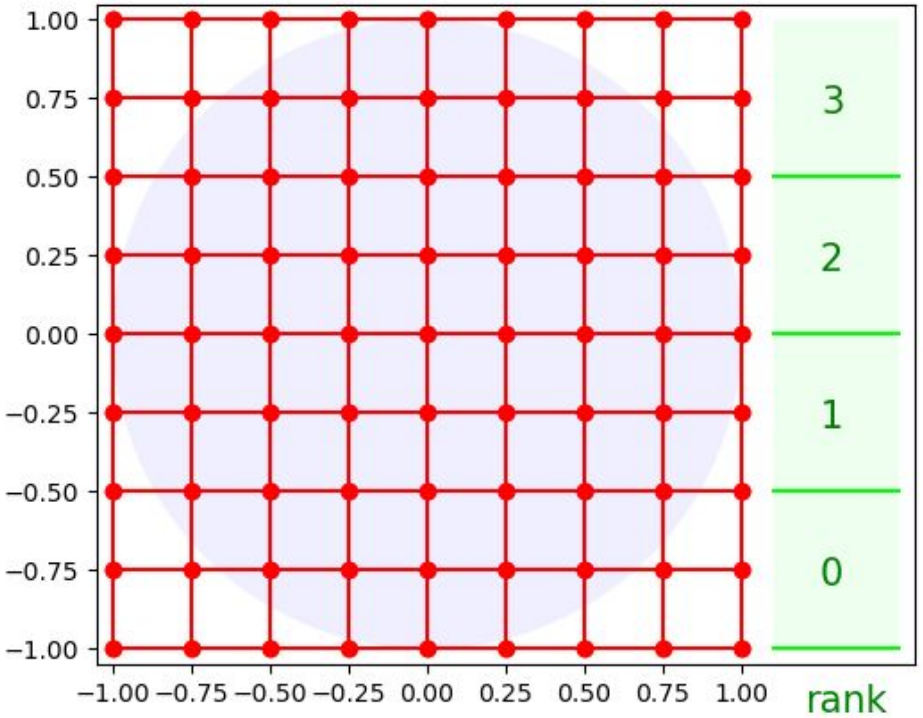
Computação Paralela / Computação Avançada

Helmut Wolters cap4 - 2022-11-02

Discretisation of real numbers in a grid

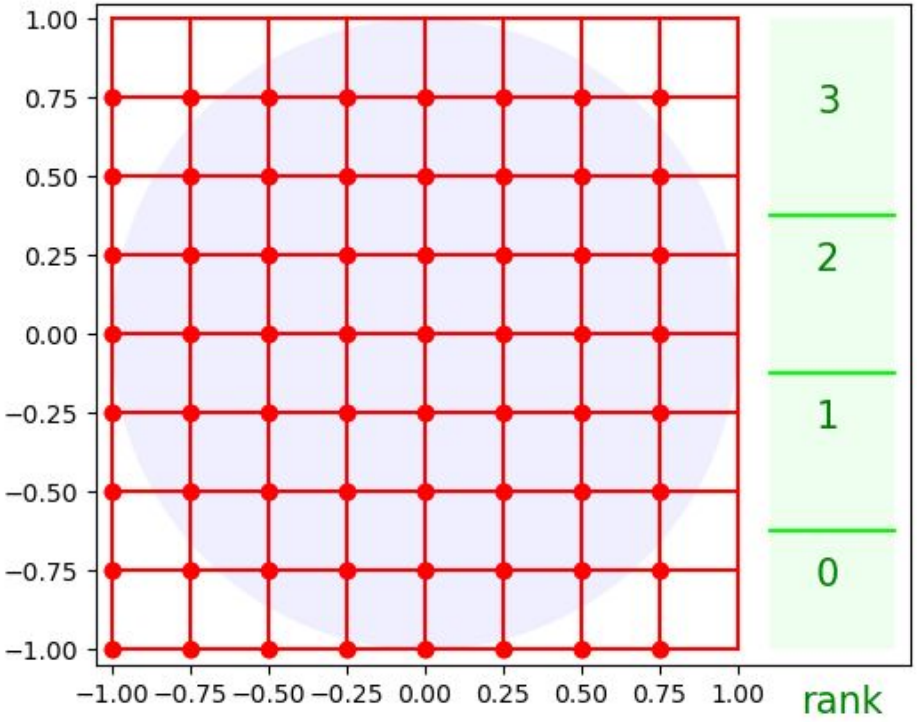


Discretisation of real numbers in a grid



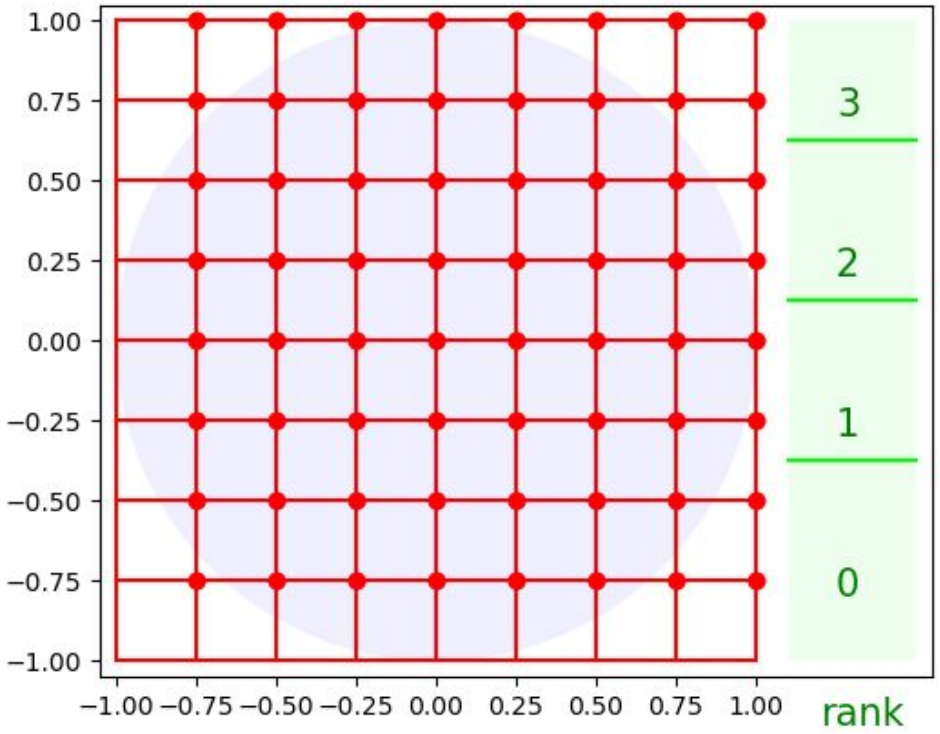
- We have an area of 1.0 x 1.0
- We want to divide it into a 8x8 grid
- This corresponds to a division into 64 0.25 x 0.25 squares
- Subdivide x in -1, -0.75, … , 0.75, 1
- **This gives 9 numbers, not 8!**
- When we subdivide the y axis for distribution over 4 parallel processes, we are likely to repeat the y values on the intervals' borders
- This would be a **systematic error**

Discretisation of real numbers in a grid



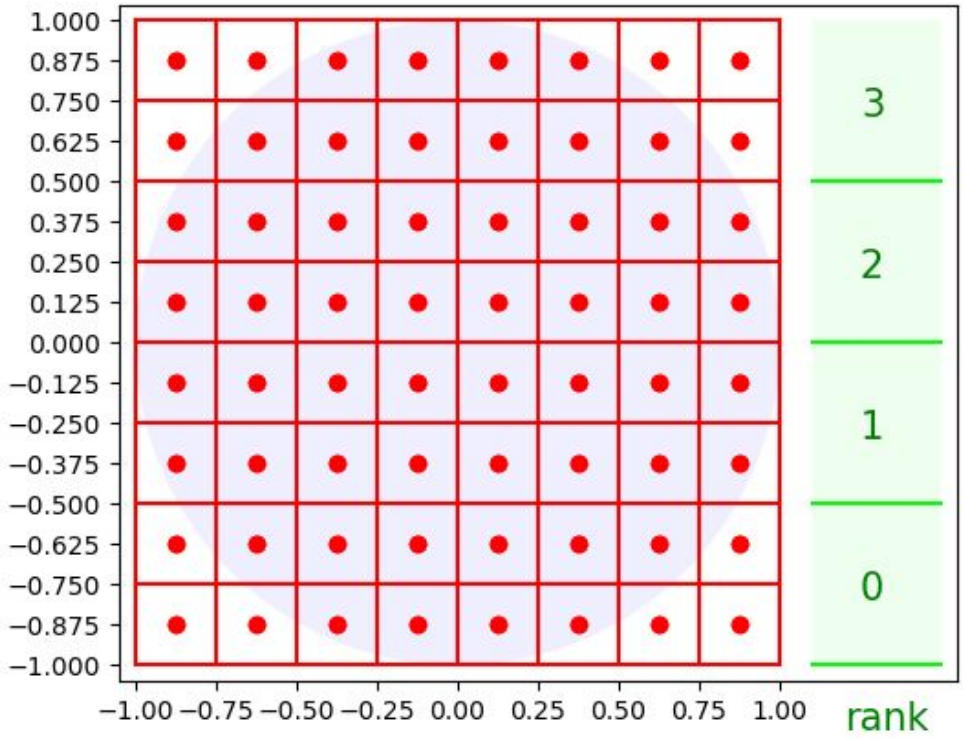
- We have an area of 1.0 x 1.0
- We want to divide it into a 8x8 grid
- This corresponds to a division into 64 0.25 x 0.25 squares
- Subdivide x in
  -1, -0.75, … , 0.75, 1
  This gives 9 numbers, not 8!
- So we take away the highest values for x and y …
- When dividing in subintervals, proceed the same way

Discretisation of real numbers in a grid



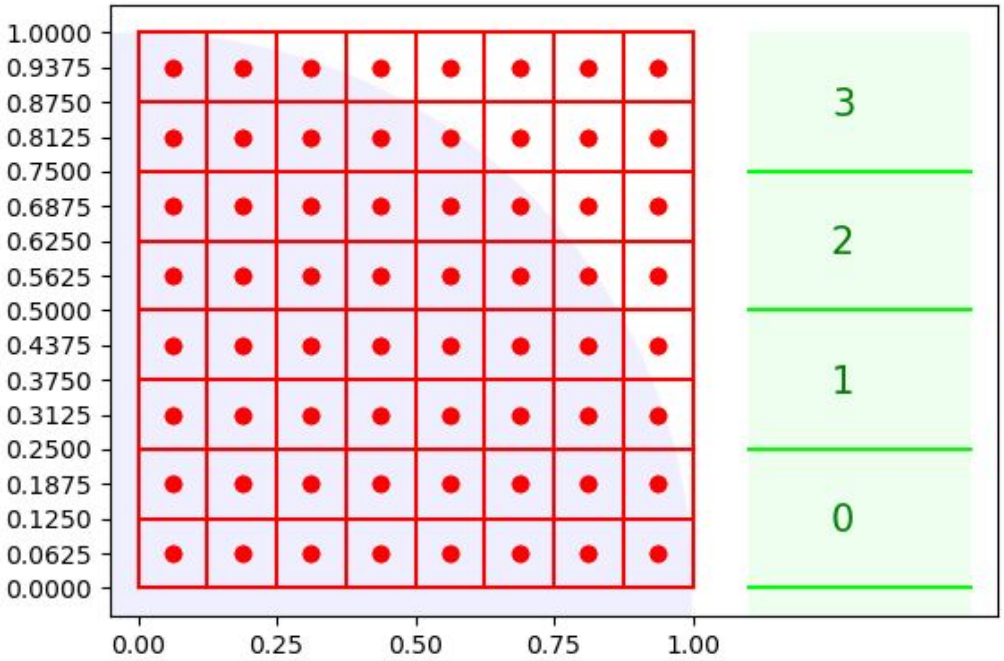
- We have an area of 1.0 x 1.0
- We want to divide it into a 8x8 grid
- This corresponds to a division into 64 0.25 x 0.25 squares
- Subdivide x in
  -1, -0.75, … , 0.75, 1
  This gives 9 numbers, not 8!
- Or we take away the lowest values for x and y …
- No repetition any more, but we still will make a **systematic error**
  shifting the values either up or down half of a square width/height

Discretisation of real numbers in a grid



- We have an area of 1.0 x 1.0
- We want to divide it into a 8x8 grid
- This corresponds to a division into 64 0.25 x 0.25 squares
- Subdivide x in
  -1, -0.75, … , 0.75, 1
  This gives 9 numbers, not 8!
- **Correct solution:**
  The discret x and y values are the intermediate points in the intervals

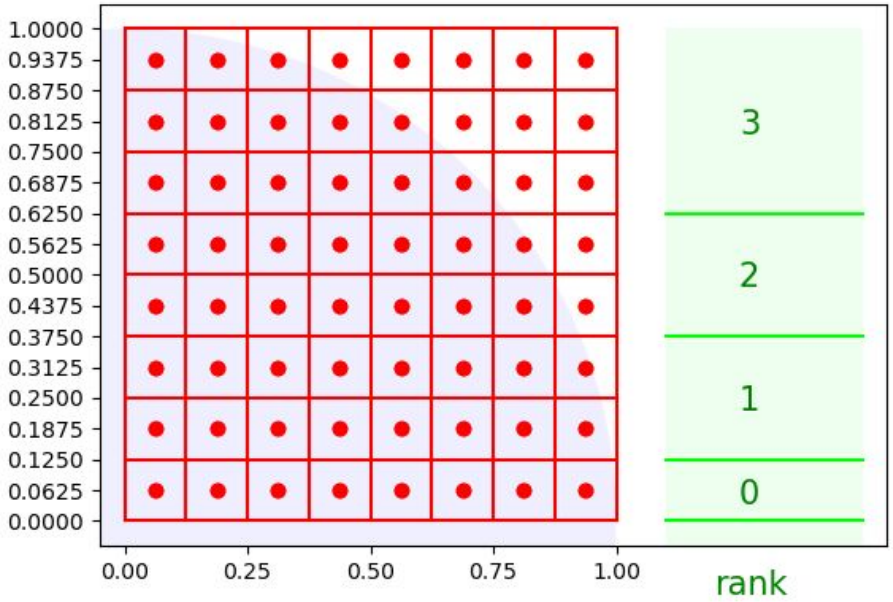
Discretisation of real numbers in a grid



- **Correct solution:**
  The discret x and y values are
  the intermediate points in the intervals
- This also makes sure that we do not repeat values when we consider only the upper right quadrant of the square and the circle

- In the “optimized” version
  (sheet 3, problem 4)
  we make one run **with small n** and **measure the time each process needs**.
- We will observe that 0 is slowest,
  1 a bit faster, 2 still a bit faster,
  and 3 much faster.
- We redistribute the share of each process, 0 gets less until 3 that gets most

Discretisation of real numbers in a grid



- In the “optimized” version (sheet 3, problem 4) we make a **first run with small n** and **measure the time each process needs**.
- We will observe that 0 is slowest, 1 a bit faster, 2 still a bit faster, and 3 much faster.
- We redistribute the share of each process, 0 gets less until 3 that gets most

- Then we run with the final n.

- How to calculate the share? Be creative, just a simple formula: Δx_p proportional to $1/\Delta t_p$

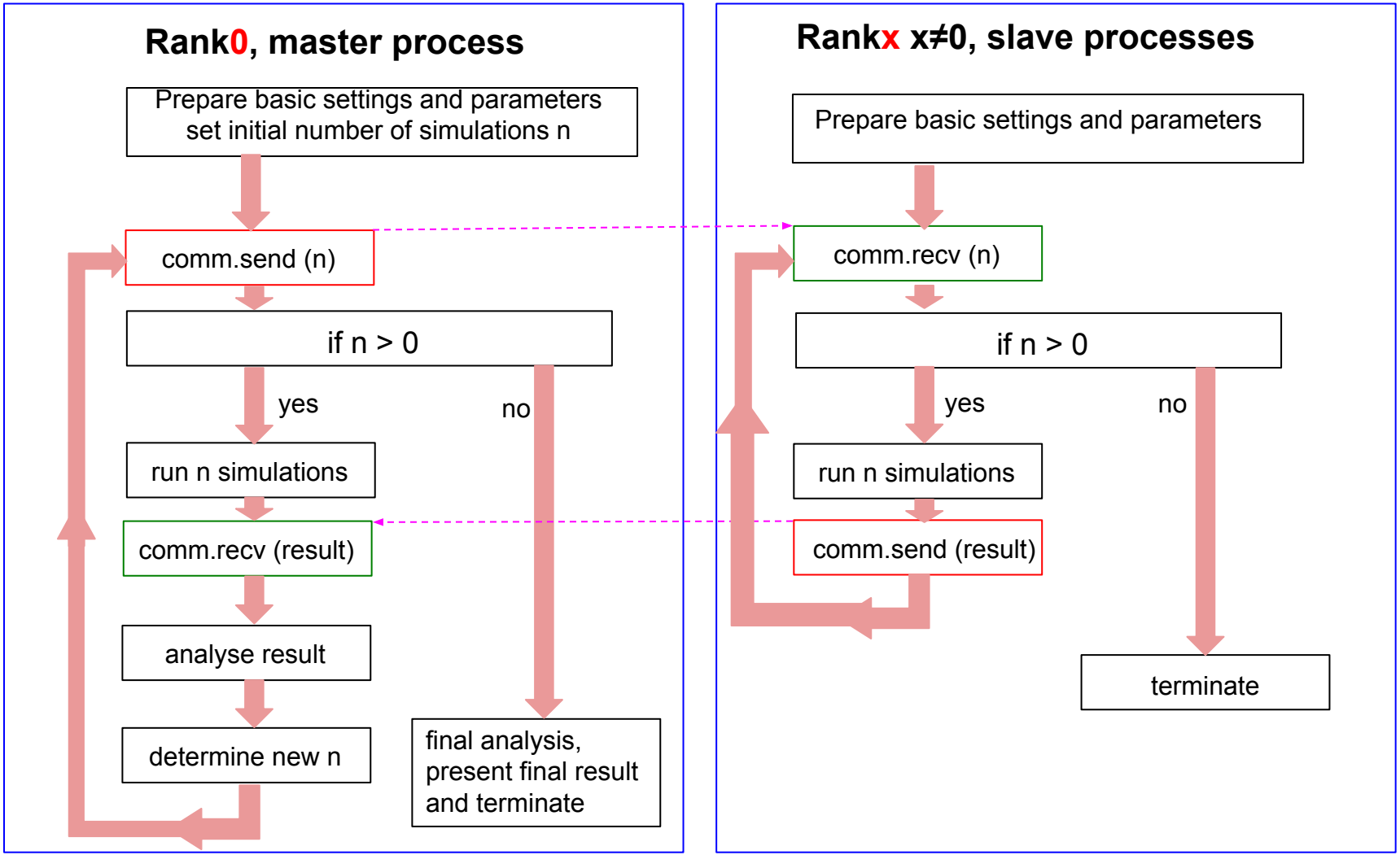
  Where, for each processor p, Δt_p is the time needed in the **first run**.

Computação Paralela / Computação Avançada

Basic flow of an iterative parallel process

- Idea: start with n simulations, distributed over m processes
- Verify quality of result
- While quality below expectations, substitute n ⇨ n' = 2*n
- When result is satisfying, stop the whole process
- Whenever possible, accumulate the results we already obtained, i.e. do not throw away what we have already calculated

- The master controls the process
- The master sends the number of simulations to be done to all other processes
- All other processes wait for this message
- When the message arrives, each process starts the simulations, (as does the master)
- All other processes send their result to the master, and wait for a new message
- The master analyses the quality of result achieved
- If not satisfying, the master doubles n and sends the new number of simulations to be done to all other processes
- etc …
- When the master sees that the analysis result is satisfying, it sends 0 to all other processes, and terminates
- When the message from the master is "0", the other processes terminate


Rank0, master process
Prepare basic settings and parameters
set initial number of simulations n
comm.send (n)
if n > 0
yes
no
run n simulations
comm.recv (result)
analyse result
determine new n
final analysis,
present final result
and terminate
Rankx x≠0, slave processes
Prepare basic settings and parameters
comm.recv (n)
if n > 0
yes
no
run n simulations
comm.send (result)
terminate

Computação Paralela / Computação Avançada

Basic flow of an iterative parallel process

- You can use the “while” loop in python to repeat the sequences
- The best approach is a “permanent” while-loop, with a break-statement inside the loop at the point we want to finish. The break-statement continues with the next statement after the loop:

```
i = 0
while True:
        if i == 10:
                break
        print (i)
        i += 1
print ('terminated.')
```

- Be careful that any message a process is waiting for in comm.recv, has a corresponding comm.send from the other process, otherwise you create a deadlock in the waiting process