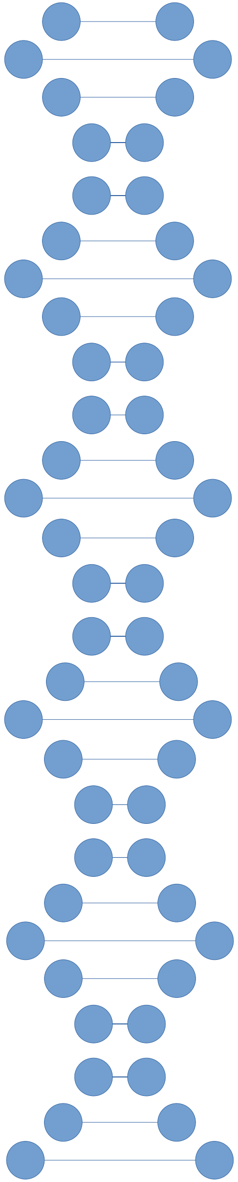
Predator and Prey Simulation in Python

Jannis Rowold, Leander Feldmann

HPC-practical 2023

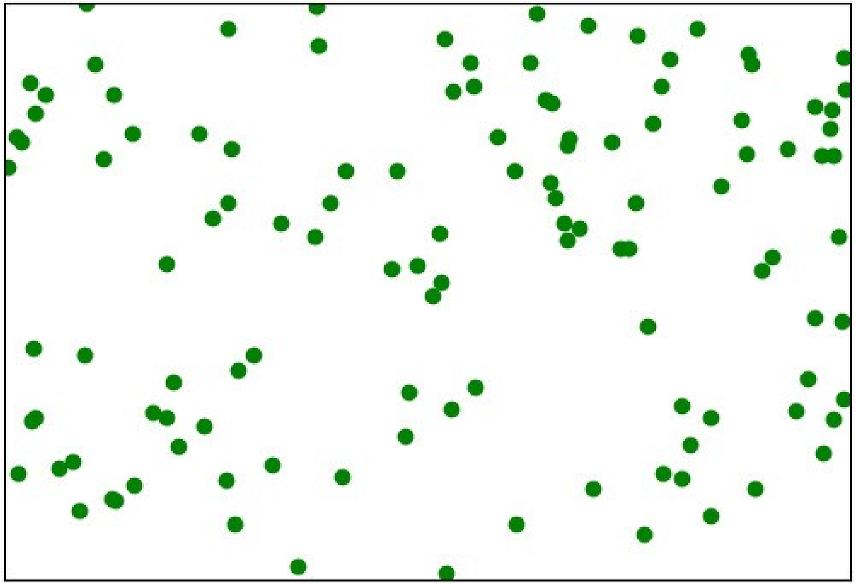


Agenda

- Idea
- Sequential approach
- First parallel approach
- Second parallel approach
- Benchmarking
- Outlook

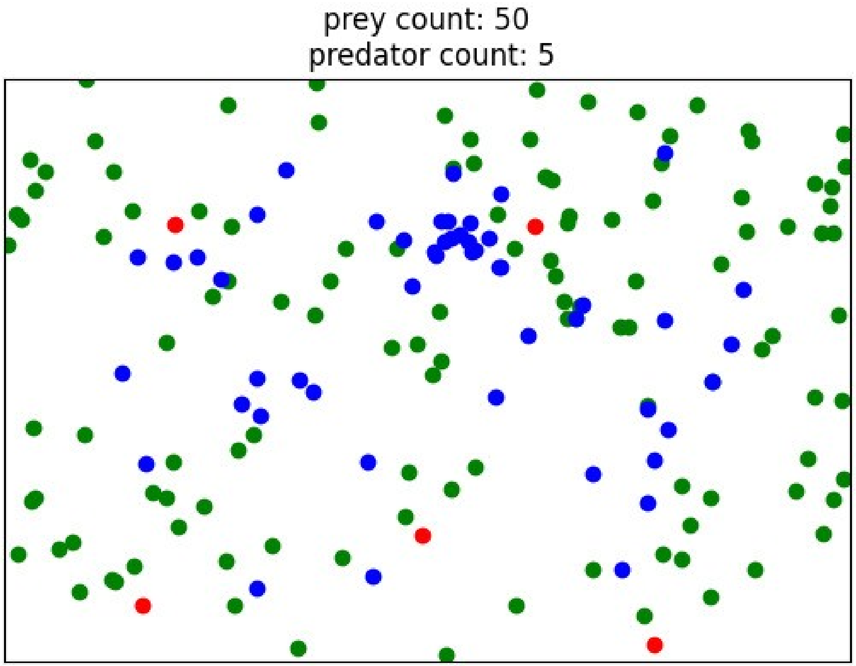
Idea

- 2D plane
- plants spawn randomly



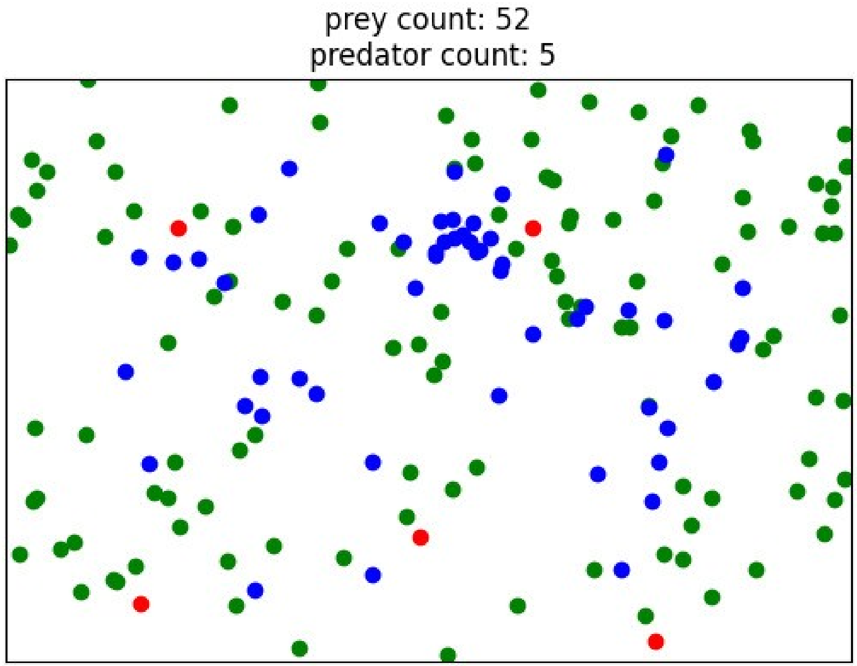
Idea

- 2D plane
- plants spawn randomly
- predator and prey spawn
- predator and prey move around



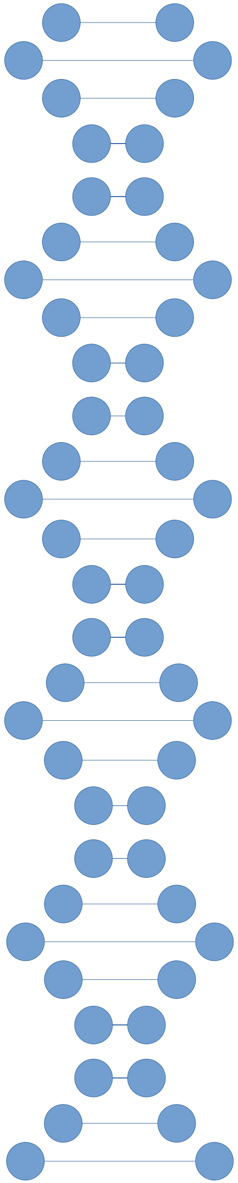
Idea

- 2D plane
- plants spawn randomly
- predator and prey spawn
- predator and prey move around



Idea - Life cycle

- Life cycle based around energy
- 0 Energy → Death
- High energy → Reproduction
- Energy is gained trough eating
- Energy decreases through moving



Idea - Creatures

Prey

- Eats plants
- Flees from predators
- Spreads out

Predator

- Eats preys
- Spreads out
- Moves towards plants

Every creature has unique speed and sensing distance

Sequential Approach

- Init
- Loop
  - creatures scan their surroundings
  - creatures move depending on their surroundings
  - creatures near food eat
  - high energy creatures reproduce
  - low energy creatures die

Sequential approach - Problems

- Naive sensing function is very time consuming
- Every distance between every creature and every other object gets calculated

```
for creature:
    calculate distance to everything
    check which objects are in sensing distance
```

- Idea: split the workload

First parallel approach

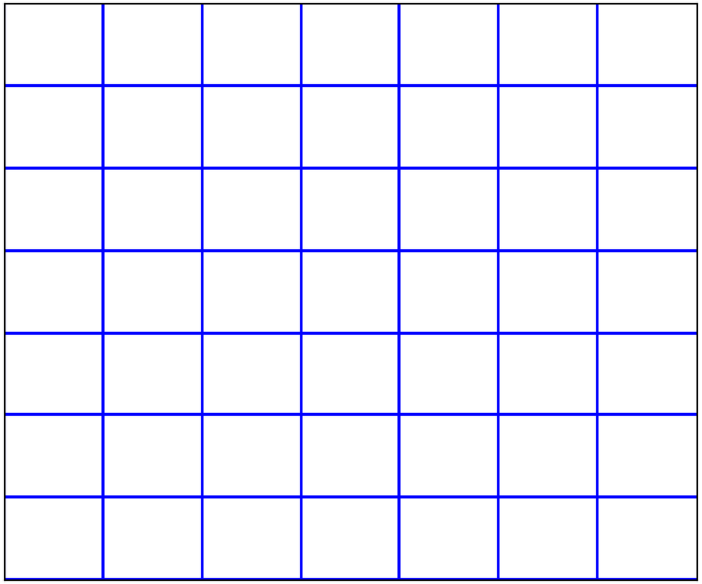
- Parallelize the calculation of the sensing distances
- Distribute the creatures on the processes evenly
- Every process has 1/n th of the calculations
- Main process splits, gathers creatures and visualize
- Worker processes calculate a subset
- No shared memory

First parallel approach - Problems

- Sensing still needs biggest portion of calculation time
- Most calculated distances are discarded right after their calculation
- Idea: minimize those calculations

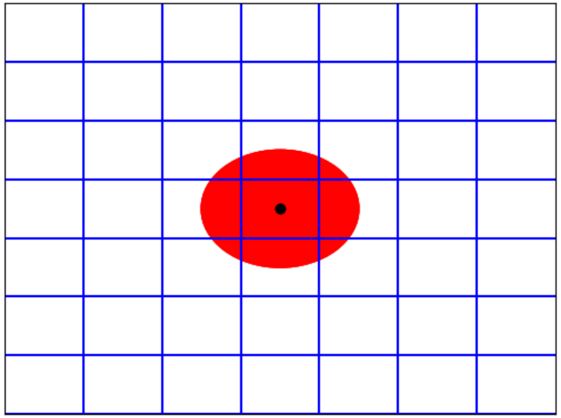
Second parallel approach

- Idea: divide plane in squares with width larger or equal max(sense)



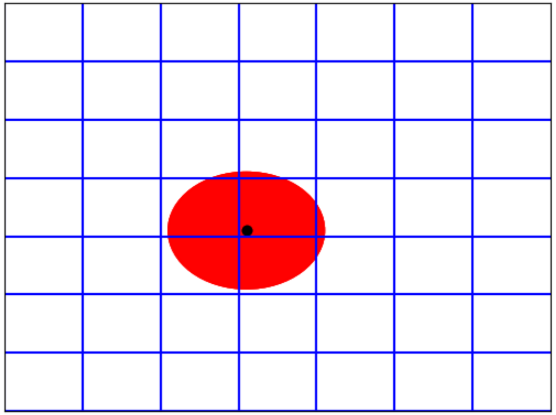
Second parallel approach

- Idea: divide plane in squares with width larger or equal max(sense)
- Sensing radius for all creatures inside a square lies within itself and the bordering squares



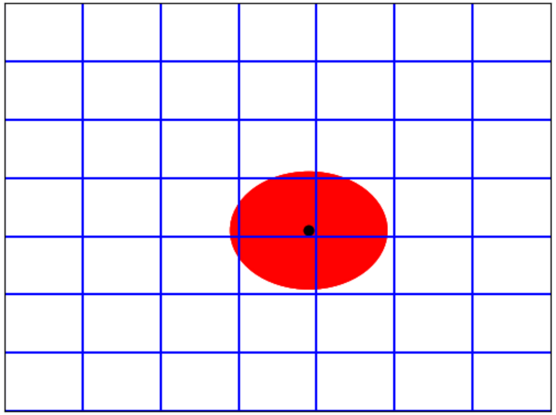
Second parallel approach

- Idea: divide plane in squares with width larger or equal max(sense)
- Sensing radius for all creatures inside a square lies within itself and the bordering squares



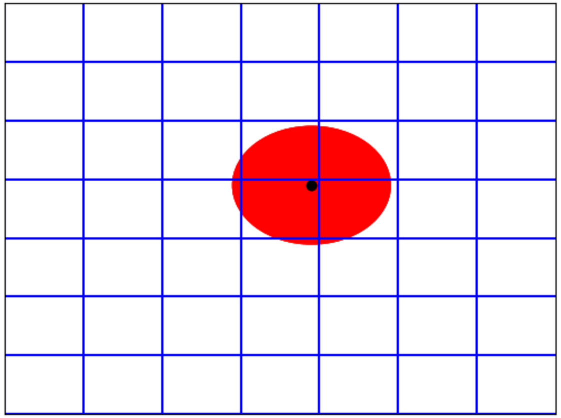
Second parallel approach

- Idea: divide plane in squares with width larger or equal max(sense)
- Sensing radius for all creatures inside a square lies within itself and the bordering squares



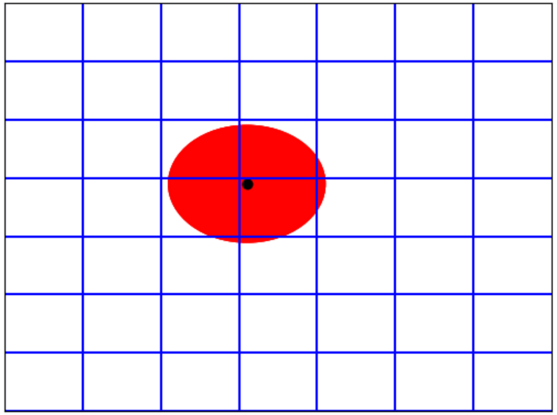
Second parallel approach

- Idea: divide plane in squares with width larger or equal max(sense)
- Sensing radius for all creatures inside a square lies within itself and the bordering squares



Second parallel approach

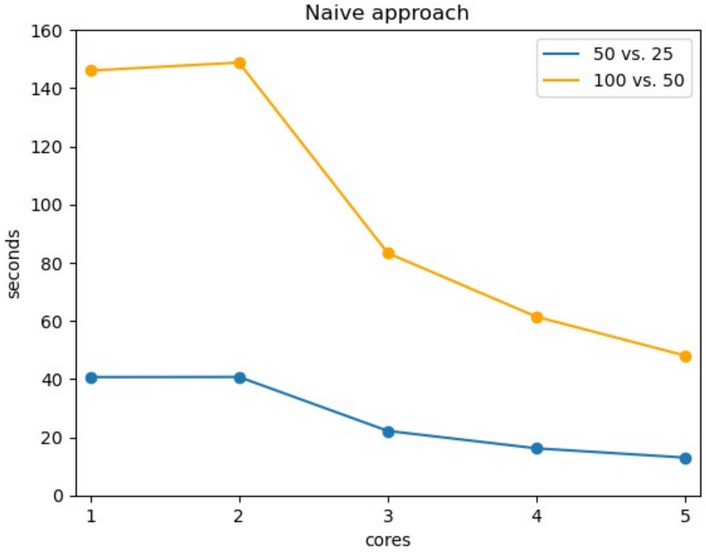
- Idea: divide plane in squares with width larger or equal max(sense)
- Sensing radius for all creatures inside a square lies within itself and the bordering squares

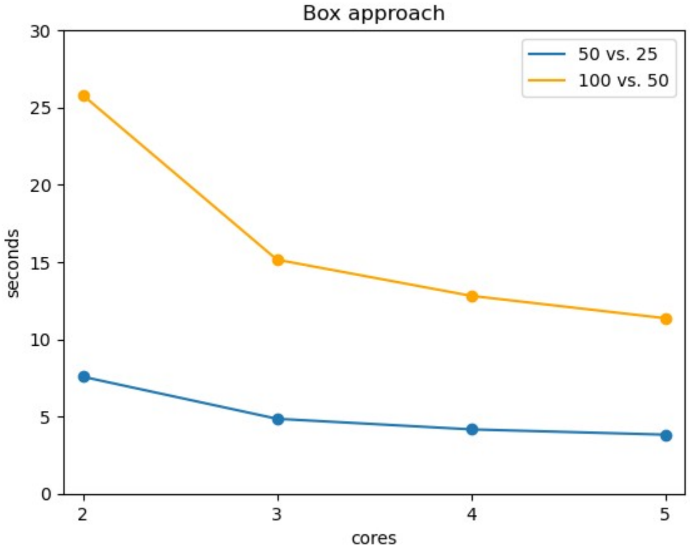


Second parallel approach

- Main process calculates boxes and allocates objects
- Worker processes receive their boxes
- For each box calculate distances and move creatures
- Main receives updated locations and energy levels
- All new data gets sorted

Benchmarking

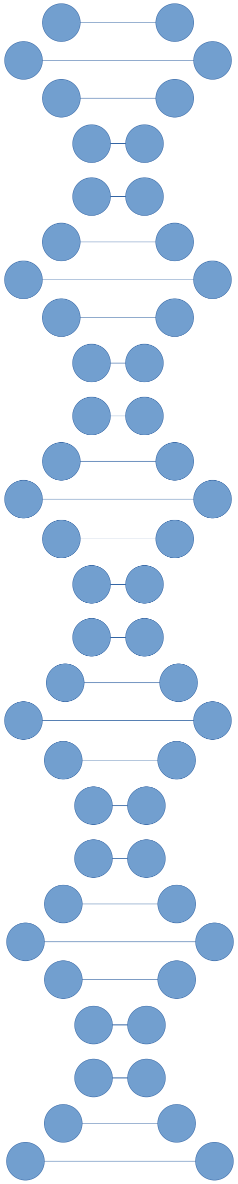




The plots show the average computation time on different amounts of cores.
The orange graph shows the results if we start with 100 prey and 50 predator.
The blue graph shows the results for a start with 50 prey and 25 predator.

Outlook - Performance

- Add reproducible outcomes for better benchmarking
- Further improve bottlenecks like distance calculation
- Reduce redundant calculations
- Try shared memory
- Use precompiled functions



Outlook - Simulation

- Add more ‘learnable’ parameters → evolution

  - size and fights

  - behavior

- Tweak starting Parameters