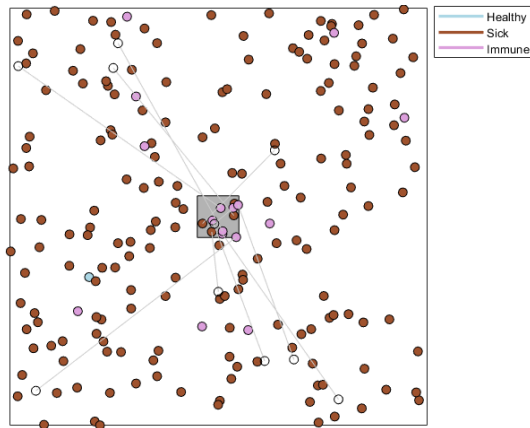# A discrete (stochastical) SIR epidemiological model

Alberto Artoni

15/03/2023

Epidemiological model[1]

Consider a discrete population of agents living in a 2D square domain.

---

[1]This exercise is inspired by models shown in
https://www.washingtonpost.com/graphics/2020/world/corona-simulator/ and
https://github.com/seismotologist/coronaVirusContagion.

Contagion model

Starting from random initial positions, the population evolves according to the following rules:

1. Each agent can be either *Susceptible*, *Infected* or *Recovered*.
2. At each timestep, agents move randomly with a given step length.
3. A prescribed percentage of agents does *social distancing*, i.e. does not move.
4. Each agent - including those who do social distancing - goes to a pub placed at the center of the domain once every fixed number of time steps.
5. A *susceptible* agent becomes infected when close to an *infected* agent.
6. An *infected* agent recovers after a fixed number of time steps.

Implementation strategy:

1. Create a sketch of the code.
2. Get a running version.
   - Each agent will become infected after meeting a susceptible agent.
   - There is no pub nor social distancing yet.
   - Create a visualization method.
3. Refine the model.
   - Add the recover dynamics.
   - Add the pub and the susceptible.
4. Improve the implementation by optimizing the code.

Exercise - Part 0

Starting from the given files, implement a C++ program that simulates the simplified model.

- Use an `enum class` to define the possible agents states.
- Exports the total number of susceptible, infected and recovered agents at each timestep to a `.csv` file (implement `Contagion::output_results`).
- Exports the positions of each agent at each timestep, use a parameter to set the frequency of the export (implement inside `Contagion::run`).

Exercise - Part 1

Implement now the dynamics of the model.

• Each `Agent` moves at random (implement a `Agent::move` method). The movement is given by a random direction (implement `Agent::generate_random_step`). If the agent moves outside the domain, the proposed generated step is changed. After $N_{max} = 1000$, if the agent is still going outside the domain, the agent stands still.

• The contagion simulation is managed by the method `Contagion::run` which at each timestep:

1. moves each agent (calls the method `Agent::move`).
2. checks the distance between the agents. If one infected agent is close to a susceptible agent, then the susceptible agent becomes infected.
3. counts the number of agents in each state.

Exercise - Part 2

- Performs the simulation employing the `<algorithm>` library and parallelize the code using the `<execution>` library where possible;
- Complete the dynamics of the model by adding the pub and the social distancing.

1. Define an enum class for each possible move performed by the agent.
2. Add the pub.

Cache aligment

Modern processors use a variety of techniques for performance. Caches is small amount of fast memory where values are "cached" in hope of reusing recently used or nearby data.

Speed (ns):	1s	10s	100s
Size (bytes):	Ks	Ms	Gs
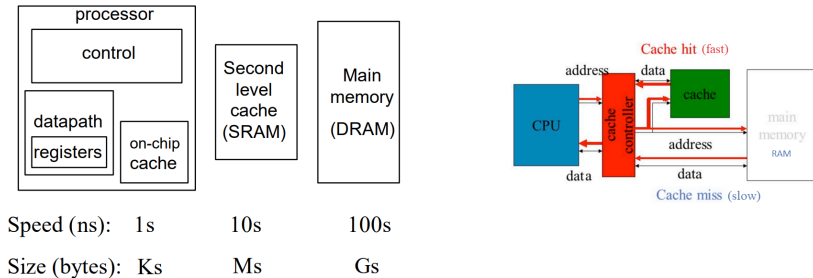
▶ Taking into account cache aligment is key for linear algebra routines
▶ Libraries like OpenBLAS (which is among the mk modules) or ATLAS are tuned at compile time to take into account hardware specific parameters (like cache size).
▶ -O3 aggressive compiler optimization option sometimes is able to do this

Exercise - Part 3

Let L1CS be the cache size of a CPU and let v be a large vector of size $K\cdot$L1CS. Consider:

```
// naive
for(size_t i = 0; i < v.size(); ++i)
  // at each iteration i we bring into the cache K different blocks of v
  // since we are accessing to the whole range
  for(size_t j = 0; j < v.size(); ++j)
    // do something with v[i] and v[j]

// partially cache optimized
for(size_t k = 0; k < K; ++k)
  for(size_t i = 0; i < v.size(); ++i)
    // at each iteration i we bring into the cache just one block of v
    // since we access only to the values from L1CS * k to L1CS * (k + 1)
    for(size_t j = L1CS * k; j < L1CS * (k + 1); ++j)
      // do something with v[i] and v[j]
```

▶ Optimize the cache alignment for the double for loop used for the radius checking step of the (sequential) simulation.

▶ Use `getconf -a | grep CACHE` to check the cache size of your PC.

▶ Confront the execution time for different number of agents between the two algorithms.

Possible extensions (homework)

1. Plot the curve of susceptible, infected and recovered using Python, you can read the .csv using the pandas module. You may need to install the matplotlib module with `pip install matplotlib`.
2. Create an animation of the positions of the agents at each timestep using Python.
3. Add social distancing.
4. Personalize agent-specific parameters, *e.g.*
   - by random generation, or
   - by reading multiple pairs of {agent index, parameter value} for each parameter from file;
5. Take birth and mortality rates into account.
6. Extend the model to 3D domains (possibly using templates).