The Journal of Open Source Software

cellular_raza: Agent-based modelling of cellular systems from a clean slate

Jonas Pleyer [1] and **Christian Fleck** [1]

1 Freiburg Center for Data-Analysis and Modelling

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

`cellular_raza` is a library that allows users to define fully-customized cellular agents in order to run numerical simulations. It formulates simulation aspects in the form of rust traits. The cellular agents and simulation domain implement a subset of these simulation aspects and `cellular_raza` provides generic methods to numerically solve the system and store results. It also comes with predefined building blocks for agents and their physical domain to quickly construct new simulations bottom-up. Furthermore, `cellular_raza` has been used with the `pyo3` and `maturin` packages to create python bindings and can act as a numerical backend to a python package.

Statement of need

Agent-based models are common in cellular biology and many tools have been developed so far to asses specific questions in specialized fields (Pleyer & Fleck, 2023). While these tools have proven to be effective for targeted research questions, they often lack the ability to be applied for multiple distinct use-cases in a more generic context. However, core functionalities such as numerical solvers, storage solutions, domain decomposition methods and functions to construct these simulations could be shared between models if written in a generic fashion.

In order to combat this issue and build up models from first principles without any assumptions on the underlying complexity or abstraction level, we developed `cellular_raza`.

- *TODO CITATIONS*

State of field

Generic agent-based modelling toolkits

There exists a wide variety of many general-purpose agent-based simulation toolkits which are being actively applied in a different fields of study (Abar et al., 2017; Wilensky, 1999; **Datseries2022?**). These tools are often able to define agents bottom-up and can be a good choice if they allow for the desired cellular representation. However, they lack the explicit forethough to be applied in cellular systems and often implement global rules rather than individual-based ones. Furthermore, since they are required to solve a wider range of problems they are not able to make assumptions on the type of agent or the nature of their interactions and thus miss out on possible performance optimizations.

Cellular agent-based frameworks

In our previous efforts (Pleyer & Fleck, 2023) we have assessed the overall state of modelling toolkits for individual-based cellular simulations. In this mini-review, we focussed on agent-

The Journal of Open Source Software

based modelling frameworks, which provide a complete workflow. The inspected frameworks are all crafted for specific use-cases and may require a large amount of parameters specific to their domain of usage. These parameters are often not known in practice and are hard to determine experimentally. This creates problems for the extendability of the software and the ability to properly interpret results.

We can further reduce the number of modeling frameworks by only considering ones which provide a significant level of flexibility and customizability in their definition of cell-agents. Chaste allows to reuse individual components of their simulation code such as ODE and PDE solvers. - *TODO CITATION* Biocellion has support for different cell shapes such as spheres and cylinders but acknowledges that their current approach lacks flexibility in the subcellular description.

- *TODO CITATION*
- *TODO check which other frameworks to consider*

Underlying Assumptions and Internals

List of Simulation Aspects

Aspect	Description	Depends on
Cellular Agent		
`Position`	Spatial representation of the cell	
`Velocity`	Spatial velocity of the cell	
`Mechanics`	Calculates the next increment from given force, velocity and position.	`Position` and `Velocity`
`Interaction`	Calculates force acting between agents. Also reacts to neighbours.	`Position` and `Velocity`
`Cycle`	Changes core properties of the cell. Responsible for cell-division and death.	
`Intracellular`	Intracellular representation of the cell.	
`Reactions`	Intracellular reactions	`Intracellular`
`ReactionsExtra`	Couples intra- & extracellular reactions	`DomainReactions`
`ReactionsContact`	Models reactions between cells purely by contact	`Position`, `Intracellular`
Simulation Domain		
`Domain`	Represents the physical simulation domain.	
`DomainMechanics`	Apply boundary conditions to agents.	`Position`, `Velocity`
`DomainForce`	Apply a spatially-dependent force onto the cell.	`Mechanics`
`DomainReactions`	Calculate extracellular reactions and effects such as diffusion.	`ReactionsExtra`
Other		
`Controller`	Externally apply changes to the cells.	

Spatially Localized Interactions

One of the most fundamental assumptions within `cellular_raza` is that each and every interaction is of finite range. This means that cellular agents only interact with their nearest neighbour and close environment. Any long-ranged interactions must be the result of a collection of short-ranged interactions. This assumption enables us to split the simulation domain into chunks and process them individually although some communication is needed in order to deal with boundary conditions. In practice, this means that any interaction force should be given a cutoff. It also means that any interactions which need to be evaluated between agents should in theory scale linearly with the number of agents $\mathcal{O}(n_{\text{agents}})$.

Code Structure

`cellular_raza` consists of multiple crates working in tandem. It was designed to have clear separations between conceptual choices and implementation details. This approach allows us to have a greater amount of modularity and flexibility than regular simulation tools.

These crates act on varying levels of abstraction to yield a fully working numerical simulation. Since `cellular_raza` functions on different levels of abstraction, we try to indicate this in the table below.

crate	Abstraction Level	Purpose
`cellular_raza`	-	Bundle together functionality of all other crates.
`concepts`	High	Collection of (mainly) traits which need to be implemented to yield a full simulation.
`core`	Intermediate-High	Contains numerical solvers, storage handlers and more to actually solve a given system.
`building_blocks`	Intermediate	Predefined components of cell-agents and domains which can be put together to obtain a full simulation.
`examples`	Application	Showcases and introductions to different simulation approaches.
`benchmarks`	Application	Performance testing of various configurations.

Backends

To numerically solve a fully specified system, `cellular_raza` provides backends. The functionality offered by a backend is the most important factor in determining the workflow of the user and how a given simulation is executed. Currently, we provide the default `chili` backend but hope to extend this collection in the future. Backends may choose to purposefully restrict themselves to a subset of simulation aspects or a particular implementation in order to improve performance.

Chili

The `chili` backend is the default choice for any new simulation. It generates source code by extensively using macros and generics. Afterwards, the generated code is compiled and run.

Every backend function is implemented generically by hand. We use trait bounds to enforce correct usage of every involved type. The generated code is restricted to methods of structs and derivations of their components functionality. To obatin a fully working simulation, the `chili` backend combines these generic methods with user-provided and generated types. The `run_simulation!` macro generates code depending on which type of simulation aspect is activated by the user. By employing this combined scheme of generics and macros, we leverage the strong type-system and Rusts language-specific safety to avoid pitfalls which a purely macro-based approach would yield.

Other Backends

`cellular_raza` also comes with the `cpu_os_threads` backend which was the first backend created. It is in the midst of being deprecated and only serves for some legacy usecases. In the future, we hope to add a dedicated backend named `cara` to leverage GPU-accelerated (Graphical Processing Unit) algorithms.

Examples

All presented examples can be viewed at the showcase section of the cellular-raza.com homepage.

Cell Sorting

Cell Sorting is a naturally occurring phenomenon which drives many biological processes. *TODO CITATION* While the underlying biological reality can be quite complex, it is rather simple to describe such a system in its most basic form. The responsible principle is that the `Interaction` between cells are specific to their species. In our example, we consider two distinct species represented by soft spheres which physically attract each other at close proximity if their species is identical.

We initially place cells randomly inside a cube with reflective boundary conditions. In the final snapshot, we can clearly see the phase-separation between the different species.

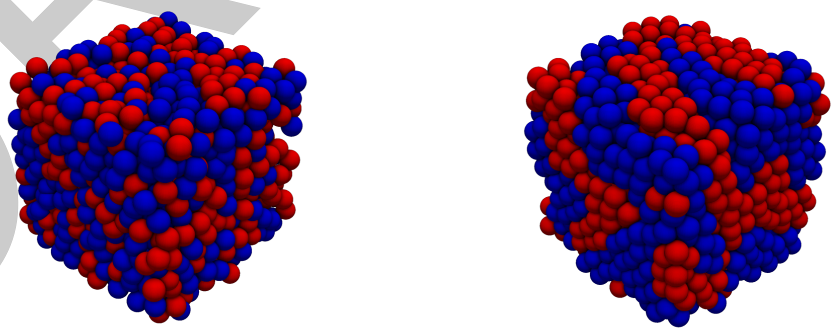

Figure 1: The initial random placement of cells reorders into a phase-separated spatial pattern.

Bacterial Rods

Many bacterial species are of elongated shape *TODO CITATION* which grows asymmetrically in the direction of elongation during the growth phase of the cell. To model this behaviour, we describe the physical `Mechanics` of one cell as a collection of multiple vertices $\vec{v}_i$ which are connected by edges. The edges are modelled as springs and their relative angle at each connecting vertex introduces a stiffening force which is proportional to the angle difference

$\alpha - 180°$. The `Interaction` of two cells is implemented via a force potential which acts between every vertex and the closest point on the other cells edges. The potential that of a soft-sphere with a short-ranged adherent force.

In addition, the cell `Cycle` introduces growth of the bacteria until it reaches a threshold and divides in the middle into two new cells. The growth is downregulated by an increasing number of neighboring cells. This can also be accomplished by the `Interaction` simulation aspect. It is an phenomenological but effective choice to model the gradual transition into the stationary phase of the bacterial colony.

Initially, the cells are placed inside the left-hand side of an elongated box with reflective boundary conditions. The cells are colored continuously from green for fast growth to blue for dormant cells. This setup is reminiscent of a mother machine continuously producing new bacteria. *TODO CITATION*

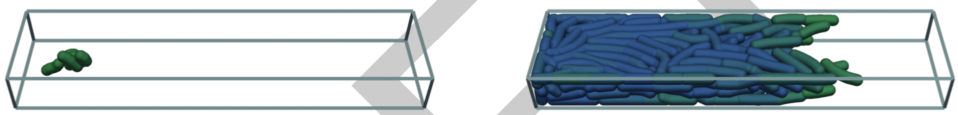

Figure 2: The bacteria extend from the initial placement in the left side towards the right side. Their elongated shape and the confined space favour the orientation facing along the growth direction.

Branching of *Bacillus Subtilis*

Spatio-temporal patterns of bacterial growth such as in *Bacillus Subtilis* have been studied for numerous years (Kawasaki et al., 1997; Matsushita et al., 1998). They are typically described by a system of PDEs (Partial Differential Equations) which contain non-spatial and spatial contributions. describing intracellular reactions and cell-cycle and spatial contributions (typically via Diffusion processes) which describe diffusion of nutrients and movement of the cells.

With `cellular_raza` we can clearly distinguish between these simulation aspects. We describe the `Mechanics` and physical `Interaction` of the cells as soft spheres. Extracellular reactions (`DomainReactions`) in the simulation domain are modeled by Diffusion which is coupled via an uptake term (`ReactionsExtra`) to the cells intracellular `Reactions`. During its life `Cycle`, the cell grows continuously and divides upon reaching a threshold.

The initial placement of the cells is inside of a centered square. From there, cells start consuming nutrients and growing outwards towards the nutrient-rich area. Cells are colored bright purple while they are actively growing and dividing while dark cells are not subject to growth anymore. The outer domain is colored by the intensity of present nutrients. A lighter color indicates that more nutrients are available while a dark color signifies a lack thereof. The two snapshots show the state after 28% of the total simulation time and at the final simulation step. The diffusivity of the nutrient and the growth rate of the bacteria are the governing criteria for the shape of the pattern.

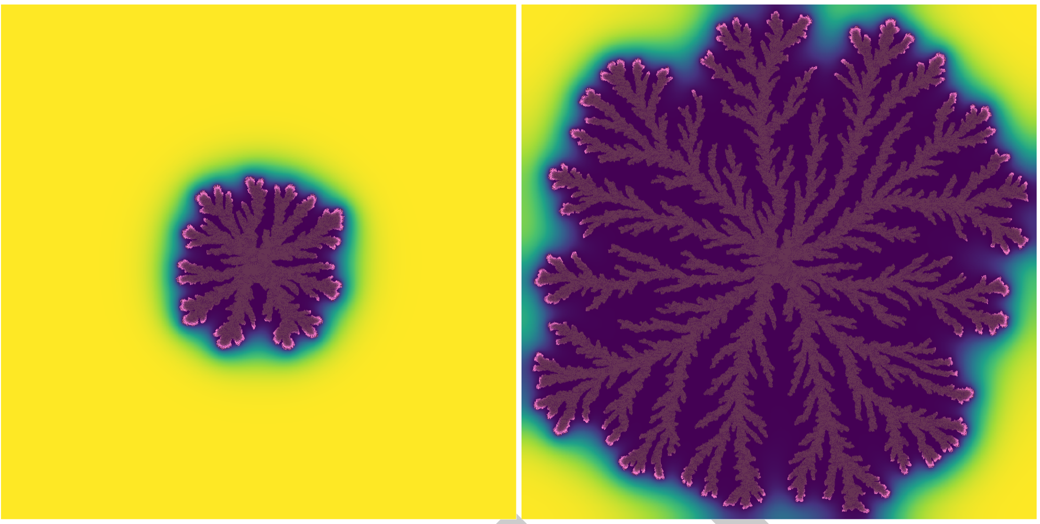

Figure 3: The bacterial colony grows outwards towards the nutrient-rich parts of the domain thus forming branches in the process.

Trichome Patterning in *Arabidopsis Thaliana*

Trichomes are hairs consisting of a single cell which can be seen developing on the surface of leaves of *Arabidopsis Thaliana*. Their specialization from regular epithelial cells is determined by a pattern which can be described by coupling the intracellular reactions of multiple cells to each other. We assume that cells are immobile but exchange the 'trichome-promoting factor TRANSPARENT TESTA GLABRA1' (TTGL) (Bouyer et al., 2008) via their connecting cell wall.

We descibe the intracellular gene reulatory network as an Ordinary Differntial Equation (ODE) via the `Intracellular` simulation aspect and couple cells to each other with the `ReactionsContact` aspect. We restrict the reactions via contact to only neighbouring cells exchange TTGL which is particularly simple to determine for a static tissue. The combined reactions represent a diffusion-driven Turing instability (Turing, 1952) which when given randomized initial values generates peaks that lead to the observed differentiation of the trichome hairs.

There is an ongoing effort (Pleyer, 2024) to use `cellular_raza` as a simulation backend while generating python bindings with the `pyo3` and `maturin` crates.

The Journal of Open Source Software

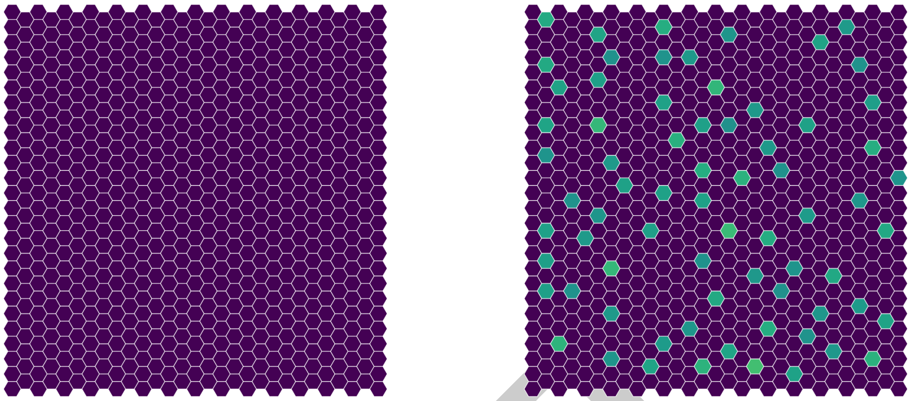

Figure 4: The reaction network produces a pattern of regular peaks that ultimately lead to the differentiation and growth of trichomes.

Performance

Multithreading

One measure of multithreaded performance is to calculate the possible theoretical speedup given by Amdahl's law (Rodgers, 1985)

$$T(n) = T_0 \frac{1}{(1-p) + \frac{p}{n}} \tag{1}$$

where n is the number of used parallel threads and p is the proportion of execution time which benefits from parallelization.

Measuring the performance of any simulation will be highly dependent on the specific cellular properties and complexity. To measure the performance of `cellular_raza`, we chose the cell sorting example which is the one containing minimal complexity of all the aforementioned example simulations. Any computational overhead which is intrinsic to `cellular_raza` and not related to the chosen example would thus be more likely to manifest in performance results. The total runtime of the simulation is of no relevance since we are only concerned with relative speedup upon using additional resources. In addition, we fixed the frequency of each processor, to circumvent power-dependent behaviour. While it is well known that other aspects such as cache-size and memory latency can have an impact on absolute performance, they should however not introduce any significant deviations in terms of relative performance scaling.

This benchmark was run on three distinct hardware configurations.

The Journal of Open Source Software

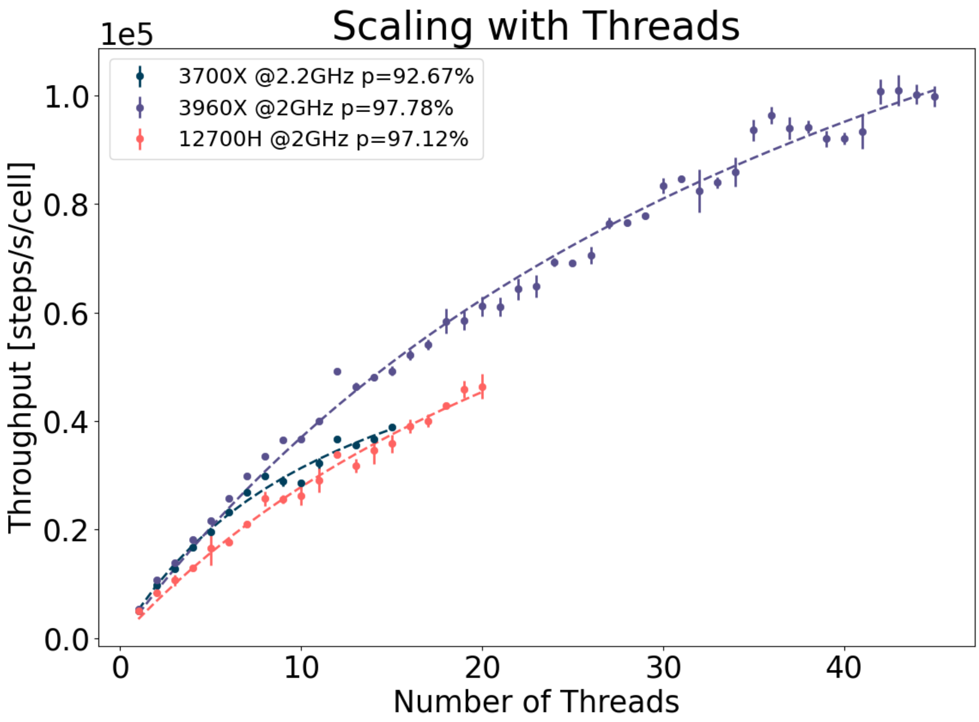

Figure 5: Amdahl's law with increasing amounts of CPU resources.

We fit equation Equation 1 and obtain the parameter p from which the theoretical maximal speedup S can be calculated via

$$S = \frac{1}{1-p} \tag{2}$$

and thus from figureFigure 5 obtain the values $S_{\text{3700X}} = 13.64$, $S_{\text{3960X}} = 45.05$ and $S_{\text{12700H}} = 34.72$.

Scaling of Simulation Size

TODO

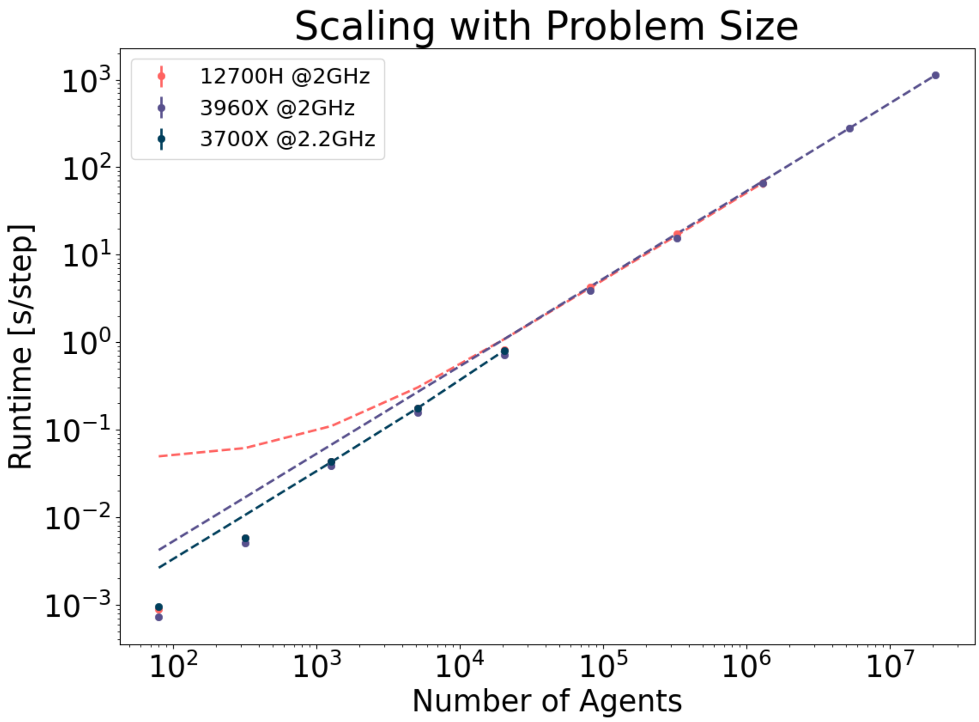

Figure 6: Scaling of the total simulation size.

Discussion

We have shown that `cellular_raza` can be applied in a wide variety of contexts. It can also serve as a numerical backend for the development of python packages. We have assessed the multithreaded performance of the implemented algorithms and shown that sufficiently large simulations can be efficiently parallelized on various machines. The underlying assumptions predict a linear growth in computational demand with linearly growing problem size which has been confirmed by our analysis.

Acknowledgements

References

Abar, S., Theodoropoulos, G. K., Lemarinier, P., & O'Hare, G. M. P. (2017). Agent based modelling and simulation tools: A review of the state-of-art software. *Computer Science Review*, *24*, 13–33. https://doi.org/10.1016/j.cosrev.2017.03.001

Bouyer, D., Geier, F., Kragler, F., Schnittger, A., Pesch, M., Wester, K., Balkunde, R., Timmer, J., Fleck, C., & Hülskamp, M. (2008). Two-dimensional patterning by a trapping/depletion mechanism: The role of TTG1 and GL3 in arabidopsis trichome formation. *PLoS Biology*, *6*(6), e141. https://doi.org/10.1371/journal.pbio.0060141

Kawasaki, K., Mochizuki, A., Matsushita, M., Umeda, T., & Shigesada, N. (1997). *Modeling Spatio-Temporal Patterns Generated byBacillus subtilis*. https://doi.org/10.1006/jtbi.1997.0462

Matsushita, M., Wakita, J., Itoh, H., Ràfols, I., Matsuyama, T., Sakaguchi, H., & Mimura, M. (1998). *Interface growth and pattern formation in bacterial colonies*. https://doi.org/10.1016/S0378-4371(97)00511-6

Pleyer, J. (2024). *Jonaspleyer/cr_trichome*. https://github.com/jonaspleyer/cr_trichome

Pleyer, J., & Fleck, C. (2023). Agent-based models in cellular systems. *Frontiers in Physics*, *10*. https://doi.org/10.3389/fphy.2022.968409

Rodgers, D. P. (1985). Improvements in multiprocessor system design. *ACM SIGARCH Computer Architecture News*, *13*(3), 225–231. https://doi.org/10.1145/327070.327215

Turing, A. M. (1952). The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, *237*(641), 37–72. https://doi.org/10.1098/rstb.1952.0012

Wilensky, U. (1999). *NetLogo* [Http://ccl.northwestern.edu/netlogo/]. Center for Connected Learning; Computer-Based Modeling. http://ccl.northwestern.edu/netlogo/