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Published: unpublished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

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 in their specialized field, they often lack the ability to be applied in a more generic context. In order to combat this issue and build up models from first principles without any assumptions on the complexity or underlying abstraction level of the model, we developed `cellular_raza`.

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). 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.

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 modelling frameworks, which provide a complete workflow. The resulting frameworks are all crafted for specific use-cases and thus lack extendability and may require a large amount of parameters specific to their domain of usage but which are often not known in practice. This creates a cost in terms of parameters and the ability to interpret results.

We can further reduce this number by only investigating frameworks 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. Biocellion has support for different cell shapes such as spheres and cylinders but acknowledge that their current approach lacks flexibility in subcellular description.

The Journal of Open Source Software

Internals

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.
`benchmarke`	Application	Performance testing of various configurations.

Backends

To numerically solve a fully specified system, `cellular_raza` provides backends. The chili backend is the default backend while the cpu-os-threads backend was the first backend which is being phased out gradually at the moment.

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.

Chili

The `chili` backend 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 structs and derivations of their components functionality. To obatin a fully working simulation, the `chili` backend combines these generic methods, user-provided and generated types. By employing this scheme, 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 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` for solving on the GPU (Graphical Processing Unit).

Underlying Assumptions

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 $\mathcal{O}(n_{\text{agents}})$.

Examples

Cell Sorting

Cell Sorting is a naturally occurring phenomenon which drives many biological processes. While the underlying biological reality can be quite complex, it is rather simple to describe such a system in its most basic form. The underlying principle is that interactions between cells are specific.

Mathematical Description

We assume that cells are spherical objects which interact via force potentials.

$$\sigma = \frac{r}{R_i + R_j} \tag{1}$$

$$V(r) = V_0 \left(\frac{1}{3\sigma^3} - \frac{1}{\sigma} \right) \tag{2}$$

The values R_i, R_j are the radii of the cells ($i \neq j$) interacting with each other. For simplification, we can assume that they are identical $R_i = R_j = R$.

Furthermore, we assume that the equation of motion is given by

$$\partial_t^2 x = F - \lambda \partial_t x \tag{3}$$

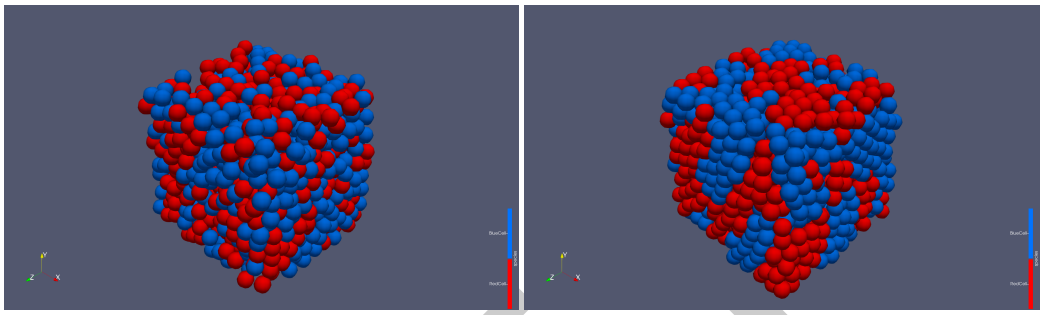
where the first term is the usual force term $F = -\nabla V$ obtained by differentiating the given potential and the second term is a damping term which arises due to the cells being immersed inside a viscuous fluid.

{{< callout type="info" >}} Note that we opted to omit the mass factor on the left-hand side of the previous equation. This means, that units of V_0 and λ are changing and they incorporate this property. {{< /callout >}}

We can assume that interactions between cells are restricted to close ranges and thus enforce a cutoff ξ for the interaction where the resulting force is identical to zero.

$$V(r) = \begin{cases} V_0 \left(\frac{1}{3\sigma^3} - \frac{1}{\sigma} \right) & \text{if } r \leq \xi \\ 0 & \text{else} \end{cases} \tag{4}$$

We further assume that cells of different species do not attract each other. To describe this behaviour, we set $V(r) = 0$ when $r > R_i + R_j$ and both cells have distince species type.



Bacterial Branching

To model the spatial mechanics of elongated bacteria, we represent them as a collection of auxiliary vertices $\{\vec{v}_i\}$ which are connected by springs in ascending order. Furthermore, we assume that the cells are flexible described by their stiffness property. A force $\vec{F}$ interacting between cellular agents determines the radius (thickness) of the rods and an attractive component can model adhesion between cells.

Mechanics

In principle we can assign individual lengths $\{l_i\}$ and strengths $\{\gamma\}_i$ to each spring. The internal force acting on vertex $\vec{v}_i$ can be divided into 2 contributions coming from the 2 springs pulling on it. In the case when $i = 0, N_{\text{vertices}}$, this is reduced to only one internal component. We denote with $\vec{c}_i$ the connection between two vertices

$$\vec{c}_i = \vec{v}_i - \vec{v}_{i-1} \tag{5}$$

and can write down the resulting force

$$\vec{F}_{i,\text{springs}} = -\gamma_i \left(1 - \frac{l_i}{|\vec{c}_i|} \right) \vec{c}_i \tag{6}$$

$$+ \gamma_{i+1} \left(1 - \frac{l_{i+1}}{|\vec{c}_{i+1}|} \right) \vec{c}_{i+1} \tag{7}$$

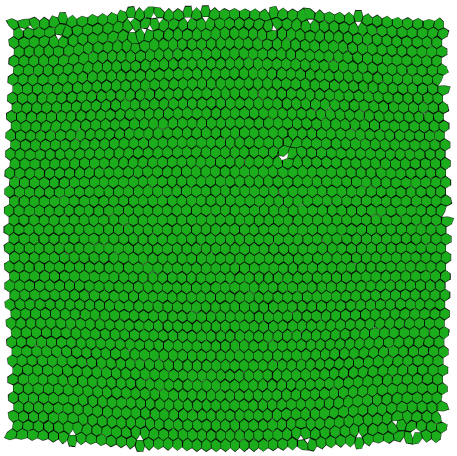
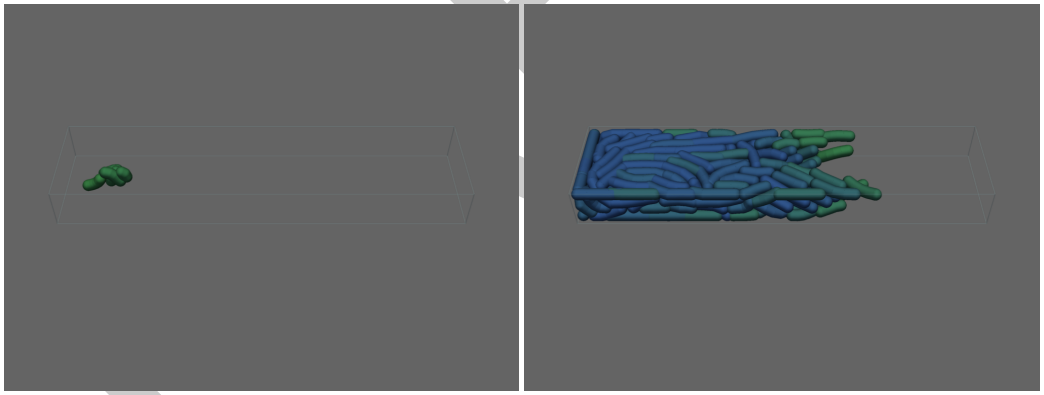


Figure 1: Free Vertex Model

Bacterial Rods



Performance

Multithreading

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

$$T(n) = T_0 \frac{1}{(1-p) + \frac{p}{n}} \tag{8}$$

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.

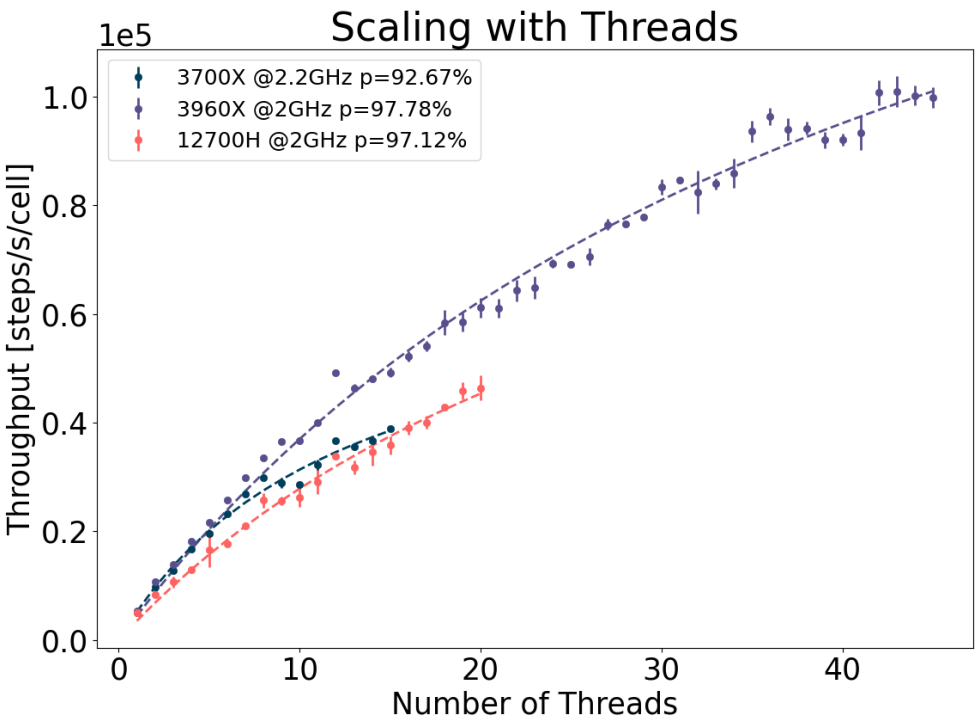


Figure 2: Amdahl's law with increasing amounts of CPU resources.

We fit equation Equation 8 and obtain the parameter p from which the theoretical maximal speedup S can be calculated via

$$S = \frac{1}{1-p} \tag{9}$$

and thus from figureFigure 2 obtain the values $S_{3700X} = 13.64$, $S_{3960X} = 45.05$ and $S_{12700H} = 34.72$.

Scaling of Simulation Size

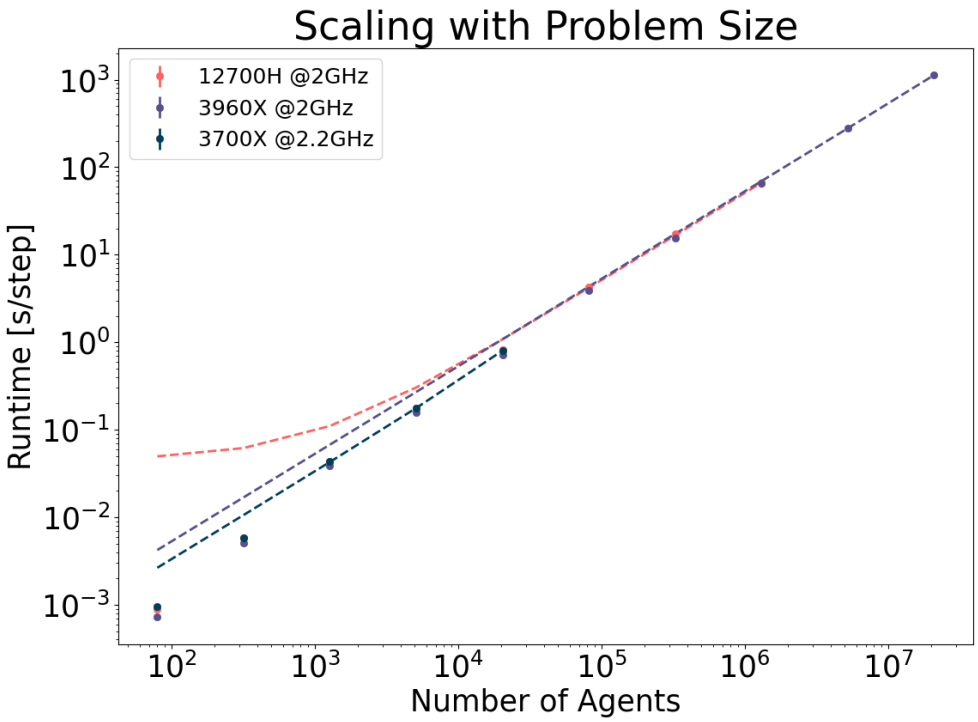


Figure 3: Scaling of the total simulation size.

Discussion

Citations

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

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