Bioinformatics
doi.10.1093/bioinformatics/xxxxxx
Advance Access Publication Date: Day Month Year
Manuscript Category

Systems biology

BioDynaMo: a general platform for scalable agent-based simulation

Lukas Breitwieser [1,2,*], Ahmad Hesam [1,3,*], Jean de Montigny [1], Vasileios Vavourakis [4,5], Alexandros Iosif [4], Jack Jennings [6], Marcus Kaiser [6,7,8], Marco Manca [9], Alberto Di Meglio [1], Zaid Al-Ars [3], Fons Rademakers [1], Onur Mutlu [2,*], Roman Bauer [10,*]

[1] CERN openlab, CERN, European Organization for Nuclear Research, Geneva, Switzerland
[2] ETH Zurich, Swiss Federal Institute of Technology in Zurich, Zurich, Switzerland
[3] Delft University of Technology, Delft, the Netherlands
[4] Department of Mechanical & Manufacturing Engineering, University of Cyprus, Nicosia, Cyprus
[5] Department of Medical Physics & Biomedical Engineering, University College London, London, UK
[6] School of Computing, Newcastle University, Newcastle upon Tyne, UK
[7] Department of Functional Neurosurgery, Ruijin Hospital, Shanghai Jiao Tong University School of Medicine, Shanghai, China
[8] Precision Imaging Beacon, School of Medicine, University of Nottingham, Nottingham, NG7 2UH
[9] SCimPulse Foundation, Geleen, Netherlands
[10] Department of Computer Science, University of Surrey, Guildford, UK

*To whom correspondence should be addressed.

Associate Editor: XXXXXXX

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Abstract

Motivation: Agent-based modeling is an indispensable tool for studying complex biological systems. However, existing simulators do not always take full advantage of modern hardware and often have a field-specific software design.
Results: We present a novel simulation platform called BioDynaMo that alleviates both of these problems. BioDynaMo features a general-purpose and high-performance simulation engine. We demonstrate that BioDynaMo can be used to simulate use cases in: neuroscience, oncology, and epidemiology. For each use case we validate our findings with experimental data or an analytical solution. Our performance results show that BioDynaMo performs up to three orders of magnitude faster than the state-of-the-art baseline. This improvement makes it feasible to simulate each use case with one billion agents on a single server, showcasing the potential BioDynaMo has for computational biology research.
Availability: BioDynaMo is an open-source project under the Apache 2.0 license and is available at www.biodynamo.org. Instructions to reproduce the results are available in supplementary information.
Contact: lukas.breitwieser@inf.ethz.ch, a.s.hesam@tudelft.nl, omutlu@ethz.ch, r.bauer@surrey.ac.uk
Supplementary information: Available at https://doi.org/10.5281/zenodo.4501515.

1 Introduction

Agent-based simulation is a powerful tool assisting life scientists in better understanding complex biological systems. In silico simulation is an inexpensive and efficient way to rapidly test hypotheses about the (patho)physiology of cellular populations, tissues, organs, or entire organisms (Yankeelov *et al.*, 2016; Ji *et al.*, 2017).

However, the effectiveness of such computer simulations for scientific research is often limited, mainly because of two reasons. First, after the slowing down of Moore's law (Moore, 1965) and Dennard scaling (Dennard *et al.*, 1974), hardware has become increasingly parallel and heterogeneous. Most simulators do not take full advantage of these hardware enhancements. The resulting limited computational power forces life scientists to compromise either on the resolution of the model or on simulation size (Thorne *et al.*, 2007). Second, existing simulators have often been developed with a specific use case in mind. This makes it challenging to implement the desired model, even if it deviates only slightly from its original purpose.

To help researchers tackle these two major challenges, we propose a novel open-source platform for biology dynamics modeling, BioDynaMo. We alleviate both of these problems by emphasizing performance and modularity. BioDynaMo features a high-performance simulation engine which is fully parallelized and able to offload computation to hardware accelerators. The software comprises a set of fundamental biological functions, and a flexible design that adapts to specific user requirements. Currently, BioDynaMo implements the biological model presented in Zubler and Douglas (2009), but this model can easily be extended, modified, or replaced. Hence, BioDynaMo is well-suited for simulating a wide range of biological processes including cell proliferation, migration, growth, etc.

BioDynaMo provides by design five system properties:

- **Agent-based.** The BioDynaMo project is established to support developmental simulations of biological dynamics. A good approximation for such in silico simulations is agent-based modeling (Railsback and Grimm, 2019). Agents are modeled as discrete objects that perform actions based on their current state, behavior, and the surrounding environment.
- **General purpose.** BioDynaMo is developed to become a general-purpose tool for agent-based simulation. To simulate models from various fields, BioDynaMo's software design is extensible and modular.
- **Large scale.** Biological systems contain a large number of agents. The cerebral cortex, for example, comprises approximately 16 billion neurons (Azevedo *et al.*, 2009). Biologists should not be limited by the number of agents within a simulation. Consequently, BioDynaMo is designed to take full advantage of modern hardware and use memory efficiently to scale up simulations.
- **Easily programmable.** The success of a simulator depends, among other things, on how quickly a scientist, not necessarily an expert in computer science or high-performance programming, can translate an idea into a simulation. This characteristic can be broken down into four key requirements that BioDynaMo is designed to fullfil:

  First, BioDynaMo provides a wide range of common functionalities such as visualization, plotting, parameter parsing, backups, etc. Second, BioDynaMo provides simulation primitives that minimize the programming effort necessary to build a use case. Third, as outlined in item "General purpose", BioDynaMo has a modular and extensible design. Fourth, BioDynaMo provides a coherent API and hides implementation details that are irrelevant for a computational model (e.g., details such as parallelization strategy, synchronization, load balancing, or hardware optimizations).
- **Quality assured.** BioDynaMo establishes a rigorous, test-driven development process to foster correctness, maintainability of the codebase, and reproducibility of results.

The main contribution of this paper is an open-source, high-performance, and general-purpose simulation platform for agent-based simulations. We provide the following evidence to support this claim: (i) We detail the user-facing features of BioDynaMo that enable users to build a simulation based on predefined building blocks and to define a model tailored to their needs. (ii) We present three basic use cases in the field of neuroscience, oncology, and epidemiology to demonstrate BioDynaMo's capabilities and modularity. (iii) We show that BioDynaMo can produce biologically-meaningful simulation results by validating these use cases against experimental data, or an analytical solution. (iv) We present performance data on different systems and scale each use case to one billion agents to demonstrate BioDynaMo's performance.

1.1 Prior work

The history of agent-based modeling and simulation goes well before the 1990s; however, it has seen widespread use in biological systems in the 2000s. Several simulators have been published demonstrating the importance of agent-based models in computational biology research (Tisue and Wilensky, 2004; Emonet *et al.*, 2005; Zubler and Douglas, 2009; Koene *et al.*, 2009; Richmond *et al.*, 2010; Collier and North, 2011; Lardon *et al.*, 2011; Rudge *et al.*, 2012; Mirams *et al.*, 2013; Torben-Nielsen and De Schutter, 2014; Kang *et al.*, 2014; Cytowski and Szymanska, 2014; Matyjaszkiewicz *et al.*, 2017; Ghaffarizadeh *et al.*, 2018). In this section, we compare BioDynaMo's most crucial system properties with prior work.

Large-scale model support. The authors of BioCellion (Kang *et al.*, 2014), PhysiCell (Ghaffarizadeh *et al.*, 2018), Timothy (Cytowski and Szymanska, 2014), and Repast HPC (Collier and North, 2011) recognize the necessity for efficient implementations to enable large-scale models. Although these tools can simulate a large number of agents, they do not support neural development. The NeuroMaC neuroscientific simulator (Torben-Nielsen and De Schutter, 2014) claims to be scalable, but the authors do not present performance data and present simulations with only 100 neurons. Therefore, BioDynaMo's ability to simulate large-scale neural development, which we demonstrate in the results section, is, to our knowledge, unrivaled.

General-purpose platform. Many simulators focus on a specific application area: bacterial colonies (Emonet *et al.*, 2005; Matyjaszkiewicz *et al.*, 2017; Rudge *et al.*, 2012; Lardon *et al.*, 2011), cell colonies (Kang *et al.*, 2014; Mirams *et al.*, 2013; Cytowski and Szymanska, 2014), and neural development (Zubler and Douglas, 2009; Koene *et al.*, 2009; Torben-Nielsen and De Schutter, 2014). Pronounced specialization of a simulator may prevent its capacity to adapt to different use cases or simulation scenarios. In contrast, BioDynaMo is a general-purpose platform for agent-based simulations by being both modular and extensible.

Quality assurance. Automated software testing is the foundation of a modern development workflow. Unfortunately, several simulation tools (Zubler and Douglas, 2009; Rudge *et al.*, 2012; Lardon *et al.*, 2011; Koene *et al.*, 2009; Torben-Nielsen and De Schutter, 2014; Cytowski and Szymanska, 2014) omit these tests. Mirams *et al.* (2013) recognize this shortcoming and describe a rigorous development workflow in their paper. BioDynaMo has over 280 automated tests which are continuously executed on all supported operating systems to ensure high code quality. BioDynaMo's open-source code base, tutorials, and documentation not only help users get started, but also enable validation by external examiners.

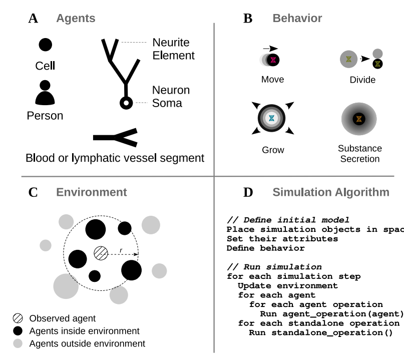
2 Design and implementation

In this section, we present the main simulation concepts of BioDynaMo and describe our approach to achieve modularity, extensibility, and high performance. We provide further information about the biological model, software quality, and features like web-based interactive notebooks, and backups in Supplementary File S1 Section 1.

2.1 Simulation concepts

BioDynaMo is implemented in the C++ programming language and supports simulations that follow an agent-based approach. Figure 1 gives an overview of BioDynaMo's main concepts, while Figure 2 illustrates its object-oriented design.

A characteristic property of agent-based simulations is the absence of a central organization unit that orchestrates the behavior of all agents. Quite to the contrary, each agent is an autonomous entity that individually determines its behavior. An agent (Figure 1A) has a 3D geometry, behavior, and environment. There is a broad spectrum of entities that can be modeled as an agent. In the results section we show examples where an agent represents a subcellular structure (neuroscience use case), a cell (oncology use case), or an entire person (epidemiology use case). Currently, BioDynaMo supports agents with cylindrical and spherical geometry. Figure 1B shows example agent behaviors such as growth factor secretion, chemotaxis, or cell division. Like genes, behaviors can be activated or inhibited. BioDynaMo achieves this by attaching them to or removing them from the corresponding agent. BioDynaMo simplifies the regulation of behaviors if new agents are created. The user can control if a behavior will be copied to a new agent or removed from the existing agent, based on the event type.



```
// Define initial model
Place simulation objects in space
Set their attributes
Define behavior

// Run simulation
for each simulation step
  Update environment
  for each agent
    for each agent operation
      Run agent_operation(agent)
  for each standalone operation
    Run standalone_operation()
```

Fig. 1. Simulation concepts. Overview of the high-level simulation concepts of BioDynaMo. Agents (A) have their own geometry, behavior (B), and environment (C). (B) Agent behavior is defined in separate classes, which are inserted into agents. A few possible examples for agents and behaviors are displayed. The update of an agent is based on its current state and its surrounding environment. (C) The environment is determined by radius r and contains other agents or extracellular substances. The simulation algorithm (D) can be divided into two main parts: the definition of the initial model and execution of the simulation.

The *Environment* is the vicinity that the agent can interact with (Figure 1C). It comprises other agents and chemical substances in the extracellular matrix. Surrounding agents are, for example, needed to calculate mechanical interactions among agent pairs. BioDynaMo determines the environment based on a uniform grid implementation. The implementation divides the total simulation space into uniform boxes of the same size and assigns agents to boxes based on the center of mass of the agent. Hence, the agents in the environment can be obtained by iterating over the assigned box and all its surrounding boxes (27 boxes in total). The box size is chosen based on the largest agent in the simulation to ensure all mechanical interactions are taken into account.

Currently, the user defines a simulation programmatically in C++ (Figure 1D). There are two main steps involved: initialization and execution. During initialization, the user places agents in space, sets their attributes, and defines their behavior. In the execution step, the simulation engine evaluates the defined model in the simulated physical environment by executing a series of operations. We distinguish between agent operations and standalone operations (Figure 2). At a high level, an agent operation is a function that: (i) alters the state of an agent and potentially also its environment, (ii) creates a new agent, or (iii) removes an agent from the simulation. Examples for agent operations are: execute all behaviors and calculate mechanical forces. The simulation engine executes agent operations for each agent for each time step. Alternatively, standalone operations perform a specific task during one time step and are therefore only invoked once. Examples include the update of substance diffusion and the export of visualization data. Supplementary File S1 Section 1.1.3 contains more details about how operations enable multi-scale simulations.

2.2 Modularity

BioDynaMo is a simulation platform that can be used to develop simulations in various computational biology fields (e.g. neuroscience, oncology, epidemiology, etc.). Although agent-based models in these different fields may intrinsically vary, there is a set of functionalities and definitions that they have in common. These commonalities, which consist of simulation and support features, are part of the BioDynaMo core. Simulation features include the physics between cellular bodies, the diffusion of extracellular substances, and basic behavior, such as proliferation and cell death. Support features include visualization, data analysis, plotting, parameter management, simulation backups, etc. Functionalities that are field-specific are separated from the core and are bundled as a separate module. Figure 2 gives an overview of BioDynaMo's software design. de Montigny *et al.* (2021) demonstrated BioDynaMo's modularity by coupling it with another simulator to create a hybrid agent-based, continuum-based model.

Neuroscience module. The neuroscience module is an example of how to extend functionality in the core to target BioDynaMo to a specific field. The module adds two new agents `NeuronSoma` and `NeuriteElement`, and models behavior like neurite extension from the soma, neurite elongation, and neurite bifurcation. The model closely follows the principles of Cortex3D (Zubler and Douglas, 2009). Neurites are implemented as a binary tree. Each neurite element can have up to two daughter elements. The cylindrical neurite element is approximated as a spring with a point mass on its distal end. These springs are connected to each other to transmit forces along the chain of neurite elements.

User-defined components. If the desired functionality is missing, the user can create, extend, or modify agents, behaviors, operations, and other classes as shown in Figure 2. BioDynaMo's software design focuses on loosely-coupled, well-defined components. This focus not only serves the purpose of creating a clear separation of the functionalities of BioDynaMo, but, perhaps even more significantly, allows users to integrate user-defined components without significant changes to the underlying software architecture. This facilitates collaboration and the creation of an open-model library. We anticipate this library will help researchers in implementing their models more rapidly.

2.3 Performance and parallelism

BioDynaMo's performance is based on the following seven enhancements: (i) The whole simulation engine is parallelized using OpenMP (OpenMP Architecture Review Board, 2015) compiler directives. OpenMP is a good fit since BioDynaMo exploits mostly loop parallelism (see Figure 1A). (ii) To increase the maximum theoretical speedup due to parallel processing (as described by Amdahl's law (Amdahl, 1967)), we minimize the number of serial code portions in BioDynaMo. (iii) We avoid unnecessary copying of data and optimize data access patterns on machines with non-uniform memory access (NUMA) architecture. Compute nodes with multiple NUMA nodes have different memory access latencies depending on whether a thread accesses local or remote memory. Therefore, we load-balance agents and their environment on available NUMA nodes. We built an optimized iterator over all agents to minimize threads' memory accesses to non-local memory. This is necessary because OpenMP does not have

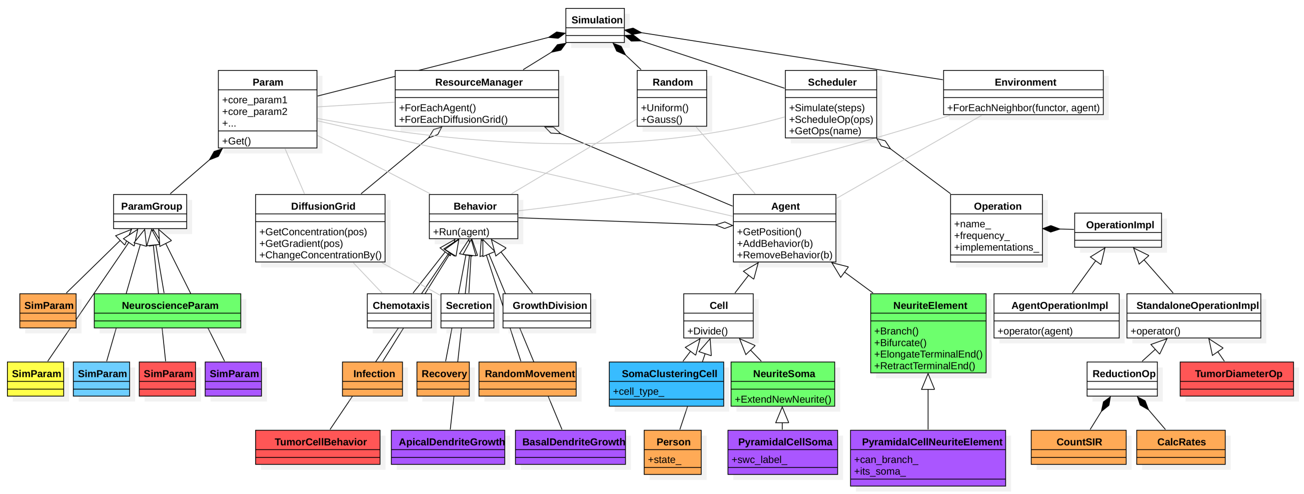


Fig. 2. Software design and modularity. Overview of selected classes and functions that are important from the users's perspective. Classes in white (BioDynaMo core) and green (BioDynaMo's neuroscience module) are part of the current BioDynaMo installation. The remaining classes illustrate how we extended BioDynaMo to implement the use cases and benchmarks shown in this paper (purple: neuroscience use case, red: oncology use case, orange: epidemiology use case, blue: soma clustering benchmark, yellow: cell proliferation benchmark). A complete list of BioDynaMo classes can be found at https://biodynamo.org/.

built-in support for such functionality. (iv) We detect stationary regions within the simulation and skip the expensive collision detection for those agents. (v) We perform just-in-time compilation to give the visualization engine ParaView direct access to Agent attributes. (vi) We develop an optimized memory allocator and concurrent hashmap. (vii) We consider offloading computations to hardware accelerator in our software design (see Figure 2). Our GPU code is implemented in NVidia CUDA and OpenCL and can be executed on graphics cards of different vendors (NVidia, AMD, or Intel). More details on BioDynaMo's performance enhancements and analyses are beyond this paper's scope, and we aim to report them in a future publication.

3 Results

This section demonstrates BioDynaMo's capacity to simulate disparate problems in systems biology with simple yet representative use cases in neuroscience, oncology, and epidemiology. Furthermore, we compare BioDynaMo's performance with an established serial neural simulator (Zubler and Douglas, 2009), analyze its scalability, and quantify the impact of GPU acceleration. For each use case we provide pseudocode for all agent behaviors, a table with model parameters, and more detailed performance results in Supplementary File S1 Section 2.

3.1 Neuroscience use case

This example illustrates the use of BioDynaMo to model neurite growth of pyramidal cells using chemical cues. Initially, a pyramidal cell, composed of a 10 μm cell body, three 0.5 μm long basal dendrites, and one 0.5 μm long apical dendrite (all of them considered here as agents), is created in 3D space. Furthermore, two artificial growth factors were initialized, following a Gaussian distribution along the z-axis. The distribution of these growth factors guided dendrite growth and remained unchanged during the simulation.

Dendritic development was dictated by a behavior defining growth direction, speed, and branching behavior for apical and basal dendrites. At each step of the simulation, the dendritic growth direction depended on the local chemical growth factor gradient, the dendrite's previous direction, and a randomly chosen direction. In addition, the dendrite's diameter tapered as it grew (shrinkage), until it reached a specified diameter, preventing it from growing any further. The weight of each element on the direction varied between apical and basal dendrites. Apical dendrites were more driven by the chemical gradient and were growing at twice the speed of basal dendrites. On the contrary, basal dendrites were more conservative in their growth direction; the weight of their previous direction was more important. Likewise, branching behavior differed between apical and basal dendrites. In addition to a higher probability of branching (0.038 and 0.006 for apical and basal respectively), apical dendrites had the possibility to branch only on the main branch of the arbor. On the contrary, basal dendrites were only ruled by a simple probability to branch at each time step.

These simple rules gave rise to a straight long apical dendrite with a simple branching pattern and more dispersed basal dendrites, as shown in Figure 3A, similar to what can be observed in real pyramidal cell morphologies as shown in Figure 3B or Spruston (2008) (Figure 1A CA1). Using our growth model, we were able to generate a large number of various realistic pyramidal cell morphologies. We used a publicly available database of real pyramidal cells coming from (Mellström *et al.*, 2016) for comparison and parameter tuning. Two measures were used to compare our simulated neurons and the 69 neurons composing the real morphologies database: the average number of branching points, and the average length of dendritic trees. No significant differences were observed between our simulated neurons and the real ones ($p < 0.001$ using a T-test for two independent samples). These results are shown in Figure 3D. The simulation of the pyramidal cell growth consisted of 361 lines of C++ code.

Figure 3C shows a large scale simulation incorporating 5000 neurons similar to the one described above, and demonstrates the potential of BioDynaMo for developmental, anatomical, and connectivity studies in the brain. This simulation contained 9 million agents. These 500 iterations correspond to approximately three weeks of pyramidal cell growth in the rat.

3.2 Oncology use case

In this section, we present a tumor spheroid simulation to replicate in vitro experiments from (Gong *et al.*, 2015). Tumor spheroid experiments are typically employed to investigate the pathophysiology of cancer, and are

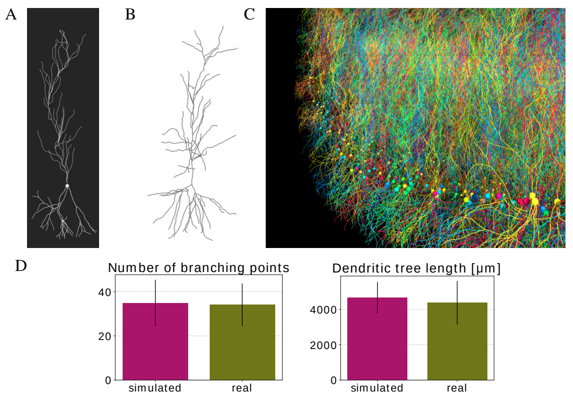


Fig. 3. Pyramidal cell simulation. (A) Example pyramidal cell simulated with BioDynaMo. (B) Real neuron (R67nr67b-CEL1.CNG) taken from (Mellström et al., 2016) and visualized with https://neuroinformatics.nl/HBP/morphology-viewer/ (C) Large-scale simulation. The model started with 5000 initial pyramidal cell bodies and contained 9 million agents after simulating 500 iterations. Simulation execution time was 46 seconds on a server with 72 CPU cores. (D) Morphology comparison between simulated neurons and experimental data from (Mellström et al., 2016). Error bars represent the standard deviation. (A,C) A video is available in Supplementary Information.

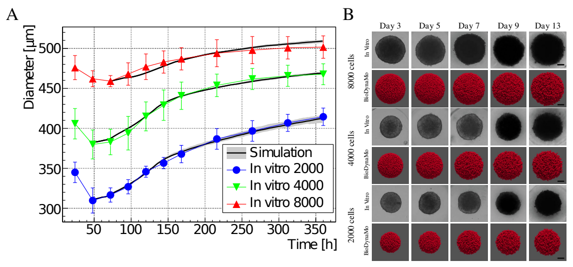


Fig. 4. Comparison between in vitro MCF-7 tumor spheroid experiments and our in silico simulations using BioDynaMo. (A) Human breast adenocarcinoma tumor spheroid (MCF-7 cell line) development during a 15 day period, where different initial cell populations were considered (see Fig 3 in (Gong et al., 2015)). Error bars denote standard deviation to the experimental data. The mean of the in silico results is shown as a solid black line with a grey band depicting minimum and maximum observed value. (B) Qualitative comparison between the microscopy images and simulation snap-shots is shown in the three boxes. Scale bars correspond to 100μm. A video is available in Supplementary Information.

also being used for pre-clinical drug screening (Nunes *et al.*, 2019). Here we considered three in vitro test cases using a breast adenocarcinoma MCF-7 cell line (Gong *et al.*, 2015) with different initial cell populations (2000, 4000, and 8000 MCF-7 cells). Our goal was to simulate the growth of this mono cell culture embedded in a collagenous (extracellular) matrix. This approach, as opposed to a free suspension one, incorporates cell-matrix interactions to mimic the tumor-host environment.

Initially, cancer cells (agents) were clustered in a spherical shape around the origin with a diameter of 310, 380, or 460 micrometers. The three-dimensional extracellular matrix (ECM) was represented in our simulations as a 8 mm^3 cube. The fundamental cellular mechanisms modeled here include cell growth, cell duplication, cell migration, and cell apoptosis. A single behavior governed all these processes. The cell growth rate was derived from the published data (Sutherland *et al.*, 1983), while cell migration (cell movement speed), cell survival, and apoptosis were fine-tuned after trial and error testing. Since the in vitro study considered the same agarose gel matrix composition among the experiments, the BioDynaMo model assumes identical parameters for the cell–matrix interactions in the simulations. Considering the homogeneous ECM properties, tumor cell migration was modeled as Brownian motion.

The in vitro experiments showed that instantaneous spheroid growth was hindered by the compression of the surrounding agarose gel matrix (see Figure 4A), owing to cell reorganization at the onset of the cancer mass implantation into the gel. As a result, the tumor spheroid diameter was initially decreasing. However, the present simulation example focuses modeling the growth of the spheroid after it had set in the agarose gel matrix. Therefore, as shown in Figure 4A, BioDynaMo simulations are set to start on day two or three.

The in vitro experiments from (Gong *et al.*, 2015) and the simulations using BioDynaMo are depicted in Figure 4. Each line plot in Figure 4A compares the mean diameter between the experiments and the simulations over time, which demonstrates the validity and accuracy of BioDynaMo. The diameter of the spheroids in the simulations were deducted from the volume of the convex hull that enclosed all cancer cells. The in vitro experiments used microscopy imaging to measure the spheroid's diameters (Gong *et al.*, 2015). Figure 4B compares snapshots of the simulated tumor spheroids (bottom row) against microscopy images of in vitro spheroids (top row) at different time points. The spheroid's morphologies between the in vitro experiments and the BioDynaMo simulations are in excellent agreement.

The example has 424 lines of C++ code, including the generation of the plot shown in Figure 4A. Running one simulation took 0.98–3.39s on a laptop and 1.24–4.16s on a server, both using one CPU core.

3.3 Epidemiology use case

This section presents an agent-based model that describes the spreading of infectious diseases between humans. The model divides the population into three groups: susceptible, infected, and recovered (SIR) agents. We compare our simulation results with the solution of the original SIR model from Kermack *et al.* (1927), which used the following three differential equation to describe the model dynamics: $dS/dt = -\beta SI/N$, $dI/dt = \beta SI/N - \gamma I$, and $dR/dt = \gamma I$. S, I, and R are the number of susceptible, infected, and recovered individuals, N is the total number of individuals, β is the mean transmission rate, and γ the recovery rate.

For our agent-based implementation (Figure 5C) we created a new agent (representing a person) that encompasses three new behaviors, and extended an operation to count the number of agents in each group (see Figure 2). Agents were randomly distributed in space and have three behaviors. Infection. A susceptible agent became infected with the infection probability if an infected agent was within the infection radius. Recovery. An infected agent recovered with the recovery probability at every time step. Random movement. All agents moved randomly in space. The absolute distance an agent may travel in every time step is limited.

In this agent-based model, the speed at which an infectious disease spreads depended on: the infection probability, the number of contacts each agent has with other agents, and the recovery rate. The number of contacts in turn depended on the infection radius, the maximum distance an agent may travel, and the density of agents in the simulation space.

We selected two infectious diseases with different characteristics to verify our model: measles and seasonal influenza. We obtained values for the basic reproduction number R_0 and recovery duration T_R from the literature (Measles: $R_0 = 12.9$, $T_R = 8$ days (Guerra *et al.*, 2017; World Health Organization, 2020), Influenza: $R_0 = 1.3$, $T_R = 4.1$ days (Chowell *et al.*, 2008)) and determined the parameters β and γ for the analytical model, based on $R_0 = \beta/\gamma$ and $\gamma = 1/T_R$. For the agent-based model we set the recovery probability to γ, and placed 2000 susceptible agents and a few infected agents randomly in a cubic space with length 100. The remaining parameters (infection radius, infection probability, and maximum movement in one time step) were determined using particle swarm optimization (Kennedy and Eberhart, 1995). Figure 5

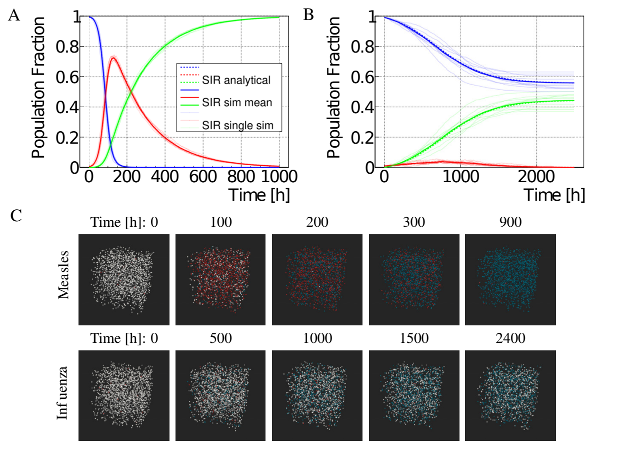


Fig. 5. Measles and seasonal influenza SIR model results. (A,B) Comparison between agent-based (solid lines) and analytical (dashed lines) model for measles (A) and seasonal influenza (B). The agent-based simulation was repeated ten times. The individual simulation results are shown as thin solid lines. The bold solid line represents the mean from all simulations. The legend is shared between the two plots. (C) Visualization of the measles and influenza model for different time steps in 3D space. Susceptible persons are shown in white, infected persons in red, and recovered persons in blue. Persons move randomly and follow the rules for infection and recovery.

shows that the agent-based model is in excellent agreement with the equation-based approach from (Kermack *et al.*, 1927) for measles and influenza.

The example has 566 lines of C++ code, including the generation of the plot shown in Figure 5. Running one simulation took 0.59–1.59s using one CPU core.

3.4 Performance

Efficient usage of computing resources is paramount for large-scale simulations with billions of agents, reduced computational costs, and low energy footprint. To this end, we quantify the performance of BioDynaMo with three simulations: cell growth and division, soma clustering, and pyramidal cell growth. These simulations have different properties and are, therefore, well suited to evaluate BioDynaMo's simulation engine under a broad set of conditions. Supplementary File S1 Section 2.2 contains more details about these benchmarks.

First, to demonstrate the performance improvements against established agent-based simulators, we compared BioDynaMo with Cortex3D (Zubler and Douglas, 2009). Cortex3D has the highest similarity in terms of the underlying biological model out of all the related works presented in Section 1.1. More specifically, BioDynaMo and Cortex3D use the same method to determine mechanical forces between agents and the same model to grow neural morphologies. This makes Cortex3D the best candidate with which to compare BioDynaMo and ensure a fair comparison. Figure 6A shows the speedup of BioDynaMo for the three simulations. We observed a significant speedup between 18 and 78×. Note that we set the number of threads available to BioDynaMo to one since Cortex3D is not parallelized. The speedup was larger, when the simulation was more dynamic or more complex.

Second, to evaluate the scalability of BioDynaMo, we measured the simulation time with an increasing number of threads. We increased the number of agents used in the comparison with Cortex3D and reduced the number of simulation timesteps to 10. Figure 6B shows the strong scaling analysis. All simulation parameters were kept constant, and the number of threads was increased from one to the number of logical cores provided by the benchmark server. The maximum speedup ranged between

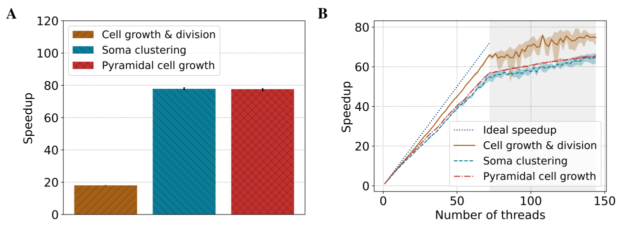


Fig. 6. BioDynaMo performance analysis. (A) Speedup of BioDynaMo compared to Cortex3D. (B) Strong scaling behavior of BioDynaMo on a server with 72 physical cores, two threads per core, and four NUMA domains. The grey area highlights hyper-threads.

65× and 75×, which corresponds to a parallel efficiency of 0.90 and 1.04. Performance improved even after all physical cores were utilized and hyper-threads were used. Hyper-threads are highlighted in gray in Figure 6B. We want to emphasize that even the pyramidal cell growth benchmark scaled well, despite the challenges of synchronization and load imbalance.

Third, we evaluated the impact of calculating the mechanical forces on the GPU using the cell growth and division, and soma clustering simulations. We excluded the pyramidal cell growth simulation because the current GPU kernel does not support cylinder geometry yet. The benchmarks were executed on System C (see Supplementary File S1 Table 4), comparing an NVidia Tesla V100 GPU with 32 CPU cores (64 threads). We observed a speedup of 1.27× for cell growth and division, and 5.04× for soma clustering. The speedup correlated with the number of collisions in the simulation. The computational intensity is directly linked with the number of collisions between agents.

In summary, in the scalability test, we observed a minimum speedup of 65×. Furthermore, we measured a minimum speedup of 18× comparing BioDynaMo with Cortex3D both using a single thread. Based on these two observations, we conclude that on System A (see Supplementary File S1 Table 4) BioDynaMo is more than three orders of magnitude faster than Cortex3D.

Based on these speedups, we executed the neuroscience, oncology, and epidemiology use cases with one billion agents. Using all 72 physical CPUs on System B (see Supplementary File S1 Table 4), we measured a runtime of 1 hour 37 minutes, 6 hours 49 minutes, and 3 hours 54 minutes, respectively. One billion agents, however, are not the limit. The maximum depends on the available memory and accepted execution duration. To be consistent across all use cases and keep our pipeline's total execution time better manageable, we decided to run these benchmarks with one billion agents. Table 5 in Supplementary File S1 shows that available memory would permit an epidemiological simulation with three billion agents. With enough memory, BioDynaMo is capable of supporting hundreds of billions of agents.

4 Discussion

This paper presented BioDynaMo, a novel open-source platform for agent-based simulations. Its modular software architecture allows researchers to implement models of distinctly different fields, of which neuroscience, oncology, and epidemiology were demonstrated in this paper. Although the implemented models follow a simplistic set of rules, the results that emerge from the simulations are prominent and highlight BioDynaMo's capabilities. We do not claim that these models are novel, but we rather want to emphasize that BioDynaMo enables scientists to (i) develop models in various computational biology fields in a modular fashion, (ii) obtain results rapidly with the parallelized execution engine, (iii) scale up the model to billions of agents on a single server, and (iv) produce results that are in agreement with validated experimental data. Although BioDynaMo is modular, we currently offer a limited number of ready-to-use simulation primitives. We are currently expanding our library of agents and behaviors to facilitate model development beyond the current capacity.

Ongoing work uses BioDynaMo to gain insights into retinal development, cryopreservation, multiscale (organ-to-cell) cancer modelling, COVID-19 spreading in closed environments, radiation-induced tissue damage, and more. Further efforts focus on accelerating drug development by replacing in vitro experiments with in silico simulations using BioDynaMo.

Our performance analysis showed improvements of up to three orders of magnitude over state-of-the-art baseline simulation software, allowing us to scale up simulations to an unprecedented number of agents. To the best of our knowledge, BioDynaMo is the first scalable simulator of neural development with cellular interactions that scales to more than one billion agents. The same principles used to model axons and dendrites in the neuroscience use case could also be applied to simulate blood and lymphatic vessels.

We envision BioDynaMo to become a valuable tool in computational biology, fostering faster and easier simulation of complex and large-scale systems, interdisciplinary collaboration, and scientific reproducibility.

Funding

This work was supported by the CERN Knowledge Transfer office [to L.B. and A.H.]; the Israeli Innovation Authority [to A.H.]; the Research Excellence Academy from the Faculty of Medical Science of the Newcastle University [to J.dM.]; the UCY StartUp Grant scheme [to V.V.]; the Medical Research Council of the United Kingdom [MR/N015037/1 to R.B., MR/T004347/1 to M.K.]; the Engineering and Physical Sciences Research Council of the UK [EP/S001433/1 to R.B., NS/A000026/1, EP/N031962/1 to M.K.]; a PhD studentship funded by Newcastle University's School of Computing [to J.J.]; the Wellcome Trust [102037 to M.K.]; the Guangci Professorship Program of Ruijin Hospital (Shanghai Jiao Tong Univ.) [to M.K.]; and by several donations by SAFARI Research Group's industrial partners including Huawei, Intel, Microsoft, and VMware [to O.M.]. The authors have declared that no competing interests exist.

Acknowledgments

We want to thank Giovanni De Toni for his work on the BioDynaMo build system.

References

Amdahl, G. M. (1967). Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), pages 483–485, New York, NY, USA. ACM.

Azevedo, F. A. C., Carvalho, L. R. B., Grinberg, L. T., Farfel, J. M., Ferretti, R. E. L., Leite, R. E. P., Jacob Filho, W., Lent, R., and Herculano-Houzel, S. (2009). Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain. *The Journal of Comparative Neurology*, **513**(5), 532–541.

Chowell, G., Miller, M. A., and Viboud, C. (2008). Seasonal influenza in the United States, France, and Australia: transmission and prospects for control. *Epidemiology & Infection*, **136**(6), 852–864. Publisher: Cambridge University Press.

Collier, N. and North, M. (2011). *Repast HPC: A Platform for Large-Scale Agent-Based Modeling*, chapter 5, pages 81–109. John Wiley & Sons, Ltd.

Cytowski, M. and Szymanska, Z. (2014). Large-Scale Parallel Simulations of 3d Cell Colony Dynamics. *Computing in Science Engineering*, **16**(5), 86–95.

de Montigny, J., Iosif, A., Breitwieser, L., Manca, M., Bauer, R., and Vavourakis, V. (2021). An in silico hybrid continuum-/agent-based procedure to modelling cancer development: Interrogating the interplay amongst glioma invasion, vascularity and necrosis. *Methods*, **185**, 94 – 104. Methods on simulation in biomedicine.

Dennard, R. H., Gaensslen, F. H., Rideout, V. L., Bassous, E., and LeBlanc, A. R. (1974). Design of ion-implanted MOSFET's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, **9**(5), 256–268.

Emonet, T., Macal, C. M., North, M. J., Wickersham, C. E., and Cluzel, P. (2005). AgentCell: a digital single-cell assay for bacterial chemotaxis. *Bioinformatics*, **21**(11), 2714–2721.

Ghaffarizadeh, A., Heiland, R., Friedman, S. H., Mumenthaler, S. M., and Macklin, P. (2018). PhysiCell: An open source physics-based cell simulator for 3-D multicellular systems. *PLOS Computational Biology*, **14**(2), e1005991.

Gong, X., Lin, C., Cheng, J., Su, J., Zhao, H., Liu, T., Wen, X., and Zhao, P. (2015). Generation of Multicellular Tumor Spheroids with Microwell-Based Agarose Scaffolds for Drug Testing. *PLOS ONE*, **10**(6), 1–18.

Guerra, F. M., Bolotin, S., Lim, G., Heffernan, J., Deeks, S. L., Li, Y., and Crowcroft, N. S. (2017). The basic reproduction number (R0) of measles: a systematic review. *The Lancet Infectious Diseases*, **17**(12), e420–e428. Publisher: Elsevier.

Ji, Z., Yan, K., Li, W., Hu, H., and Zhu, X. (2017). Mathematical and computational modeling in complex biological systems. *BioMed research international*, **2017**.

Kang, S., Kahan, S., McDermott, J., Flann, N., and Shmulevich, I. (2014). Biocellion: accelerating computer simulation of multicellular biological system models. *Bioinformatics*, **30**(21), 3101–3108.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4.

Kermack, W. O., McKendrick, A. G., and Walker, G. T. (1927). A contribution to the mathematical theory of epidemics. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, **115**(772), 700–721. Publisher: Royal Society.

Koene, R. A., Tijms, B., van Hees, P., Postma, F., de Ridder, A., Ramakers, G. J. A., van Pelt, J., and van Ooyen, A. (2009). NETMORPH: A Framework for the Stochastic Generation of Large Scale Neuronal Networks With Realistic Neuron Morphologies. *Neuroinformatics*, **7**(3), 195–210.

Lardon, L. A., Merkey, B. V., Martins, S., Dötsch, A., Picioreanu, C., Kreft, J.-U., and Smets, B. F. (2011). iDynoMiCS: next-generation individual-based modelling of biofilms. *Environmental Microbiology*, **13**(9), 2416–2434.

Matyjaszkiewicz, A., Fiore, G., Annunziata, F., Grierson, C. S., Savery, N. J., Marucci, L., and Bernardo, M. d. (2017). BSim 2.0: An Advanced Agent-Based Cell Simulator. *ACS Synthetic Biology*.

Mellström, B., Kastanauskaite, A., Knafo, S., Gonzalez, P., Dopazo, X. M., Ruiz-Nuño, A., Jefferys, J. G. R., Zhuo, M., Bliss, T. V. P., Naranjo, J. R., and DeFelipe, J. (2016). Specific cytoarchitectureal changes in hippocampal subareas in daDREAM mice. *Molecular Brain*, **9**(1).

Mirams, G. R., Arthurs, C. J., Bernabeu, M. O., Bordas, R., Cooper, J., Corrias, A., Davit, Y., Dunn, S.-J., Fletcher, A. G., Harvey, D. G., Marsh, M. E., Osborne, J. M., Pathmanathan, P., Pitt-Francis, J., Southern, J., Zemzemi, N., and Gavaghan, D. J. (2013). Chaste: An Open Source C++ Library for Computational Physiology and Biology. *PLOS Computational Biology*, **9**(3).

Moore, G. E. (1965). Cramming More Components Onto Integrated Circuits. *Electronics*, **38**.

Nunes, A., Barros, A., Costa, E., Moreira, A., and Correia, I. (2019). 3D tumor spheroids as in vitro models to mimic in vivo human solid tumors resistance to therapeutic drugs. *Biotechnology and Bioengineering*, **116**(1), 206–226.

OpenMP Architecture Review Board (2015). *OpenMP Application Program Interface Version 4.5*.

Railsback, S. F. and Grimm, V. (2019). *Agent-based and individual-based modeling: a practical introduction*. Princeton university press.

Richmond, P., Walker, D., Coakley, S., and Romano, D. (2010). High performance cellular level agent-based simulation with FLAME for the GPU. *Briefings in Bioinformatics*, **11**(3), 334–347.

Rudge, T. J., Steiner, P. J., Phillips, A., and Haseloff, J. (2012). Computational Modeling of Synthetic Microbial Biofilms. *ACS Synthetic Biology*, **1**(8), 345–352.

Spruston, N. (2008). Pyramidal neurons: dendritic structure and synaptic integration. *Nature Reviews Neuroscience*, **9**(3), 206–221.

Sutherland, R. L., Hall, R. E., and Taylor, I. W. (1983). Cell proliferation kinetics of mcf-7 human mammary carcinoma cells in culture and effects of tamoxifen on exponentially growing and plateau-phase cells. *Cancer Research*, **43**(9), 3998–4006.

Thorne, B. C., Bailey, A. M., and Peirce, S. M. (2007). Combining experiments with multi-cell agent-based modeling to study biological tissue patterning. *Briefings in Bioinformatics*, **8**(4), 245–257.

Tisue, S. and Wilensky, U. (2004). Netlogo: A simple environment for modeling complexity. In *International conference on complex systems*, volume 21, pages 16–21. Boston, MA.

Torben-Nielsen, B. and De Schutter, E. (2014). Context-aware modeling of neuronal morphologies. *Frontiers in Neuroanatomy*, **8**.

World Health Organization (2020). Measles. https://www.who.int/news-room/fact-sheets/detail/measles.

Yankeelov, T. E., An, G., Saut, O., Luebeck, E. G., Popel, A. S., Ribba, B., Vicini, P., Zhou, X., Weis, J. A., Ye, K., and Genin, G. M. (2016). Multi-scale modeling in clinical oncology: Opportunities and barriers to success. *Annals of Biomedical Engineering*, **44**(9), 2626–2641.

Zubler, F. and Douglas, R. (2009). A framework for modeling the growth and development of neurons and networks. *Frontiers in computational neuroscience*, **3**, 25.

Supplementary Materials

This document provides additional information about the design and implementation of BioDynaMo, use cases, and performance results. Furthermore, future directions and supplementary files are explained.

1 Design and implementation

1.1 Biological model

In this section, we provide more details about the biological model BioDynaMo currently implements. This model closely resembles the principles from Cortex3D (Zubler and Douglas, 2009), but can be extended or replaced easily.

1.1.1 Mechanical forces

Growing realistic cell and tissue morphologies requires the consideration of mechanical interactions between agents. Therefore, BioDynaMo examines if two agents collide with each other at every timestep. To find all possible collisions, it is sufficient to evaluate neighbors in the environment. Whenever two agents (e.g. a cell body or a neurite element) overlap, a collision occurs. If a collision is detected, the engine calculates the mechanical forces that act on them.

The mechanical force calculation between spheres and cylinders follows the same approach as the implementation in Cortex3D (Zubler and Douglas, 2009). Both in BioDynaMo and Cortex3D, the magnitude of the force is computed based on (Pattana, 2006) and comprises a repulsive and attractive component:

$$F_N = k\delta - \gamma\sqrt{r\delta} \tag{1}$$

where δ indicates the spatial overlap between the two elements, and r denotes a combined measure of the two radii:

$$r = \frac{r_1 r_2}{r_1 + r_2} \tag{2}$$

where the radii denote the radii of the interacting spheres or cylinder.

Eq 1 comprises the effects of the structural tension from the pressure between the colliding membrane segments, and the attractive force due to the cell adhesion molecules. The magnitudes of these two force components depend upon the modifiable parameters k and γ. In the current form, as in Cortex3D, these are set to 2 and 1, respectively. After the forces have been determined, the agents change their 3D location depending on the force resulting from all the mechanical interactions with neighbors. More details about the implementation of the mechanical force, including the force between neighboring neurite elements, can be found in (Zubler and Douglas, 2009).

1.1.2 Extracellular diffusion

Signaling molecules, which differentiate and regulate cells, reach their destination through diffusion (Gurdon and Bourillot, 2001). A well-studied example of this process, called morphogen gradients, is the determination of vein positions in the wing of Drosophila (Bosch *et al.*, 2017).

BioDynaMo solves the partial differential equations that model the diffusion of extracellular substances (Fick's second law) with the discrete central difference scheme (Smith *et al.*, 1985). A grid with a variable resolution is imposed on the simulation space. At each timestep, the concentration value of each grid point is updated according to

$$\begin{aligned} u^{n+1}_{i,j,k} = \big(u^n_{i,j,k} &+ \frac{\nu\Delta t}{\Delta x^2}(u^n_{i+1,j,k} - 2u^n_{i,j,k} + u^n_{i-1,j,k}) \\ &+ \frac{\nu\Delta t}{\Delta y^2}(u^n_{i,j+1,k} - 2u^n_{i,j,k} + u^n_{i,j-1,k}) \\ &+ \frac{\nu\Delta t}{\Delta z^2}(u^n_{i,j,k+1} - 2u^n_{i,j,k} + u^n_{i,j,k-1})\big) \times (1-\mu)\,, \end{aligned} \tag{3}$$

where $u^n_{i,j,k}$ is the concentration value on grid point (i, j, k) at timestep n, ν is the diffusion coefficient, μ is the decay constant, Δt is the duration of one timestep, and Δx, Δy, and Δz are the distances between grid points in the x, y, and z direction, respectively. The distances between the grid points are inversely proportional to the resolution and determine the accuracy of the solver.

In BioDynaMo, it is possible to define the diffusion behavior at the simulation boundaries. In the default implementation, which we use for our examples in the result section, substances diffuse out of the simulation space.

In some cases, it is necessary to initialize substance concentrations artificially to simplify a simulation. Therefore, BioDynaMo provides predefined substance initializers (e.g. Gaussian) and accepts user-defined functions for arbitrary distributions. We used this functionality, for example, in the pyramidal cell growth simulation.

1.1.3 Multi-scale simulation

A biological simulation has to account for dynamic mechanisms in the range from milliseconds to weeks (e.g. physical forces, reaction-diffusion processes, gene regulatory dynamics, etc.). BioDynaMo supports processes at different time scales by providing a parameter to specify the time interval between two time steps and an execution frequency for each operation. An execution frequency of one means that the corresponding operation is executed every time step. In contrast, a frequency of three would mean that the operation is executed every three time steps. This mechanism allows BioDynaMo to simulate e.g. substance diffusion and neurite growth in the same model.

1.2 Software quality assurance

Compromising on software quality can have severe consequences that can culminate in the retraction of published manuscripts (Miller, 2006). Therefore, we put tremendous effort into establishing a rigorous development workflow that follows industry best practices. Test-driven development—a practice from agile development (Beck and Gamma, 2000)—is at the core of our solution. BioDynaMo has over 280 tests distributed among unit, convergence, system, and installation tests. We monitor test coverage of our unit tests with the tool kcov (Kagstrom, 2020), and currently cover 79.8% lines of code. For each change to our repository (https://github.com/BioDynaMo/biodynamo), GitHub Actions (https://github.com/features/actions) executes the entire test suite and, upon success, updates the documentation on our website. Installation tests are executed on each supported operating system and ensure that all demo simulations run on a default system.

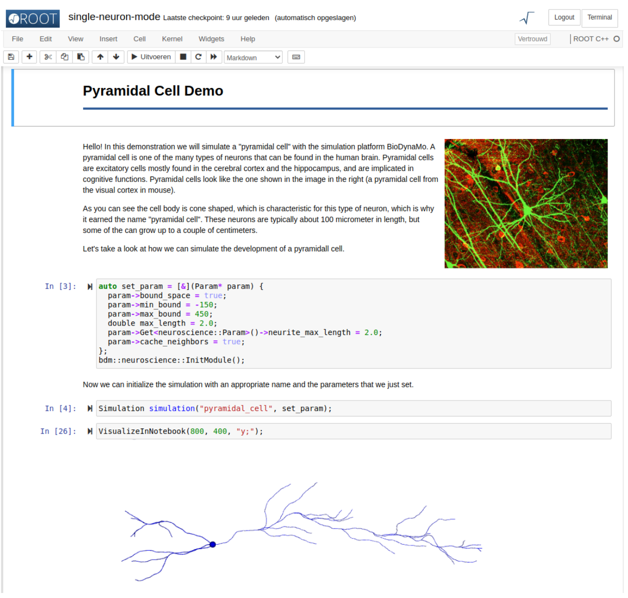
1.3 Visualization

BioDynaMo currently uses ParaView (Ahrens *et al.*, 2005) as a visualization engine. There are two visualization modes, which we refer to as live mode and export mode. With live mode, the simulation can be visualized during runtime, whereas with export mode, the visualization state is exported to file and can only be visualized post-simulation. Live mode is a convenient approach to debug a simulation visually while it is executed. However, this can slow down the simulation considerably if used continuously. In export mode, the visualization state can be loaded by the visualization package for post-simulation processing (slicing, clipping, rendering, animating, etc.). BioDynaMo can visualize substance concentrations and gradients (see Figure 2), and the geometry of the supported agents.

Furthermore, it is possible to export any agent's data members. This information can then be used as input to ParaView filters, e.g., to highlight elements based on a specific property. The export of additional data members was used in Figure 2, for example, to color cells by their cell type.

1.4 BioDynaMo notebooks

Jupyter notebooks (Kluyver *et al.*, 2016) is a widely used web application to quickly prototype or demonstrate features of a software library. With notebooks, it is possible to easily create a website with inline code snippets that can be executed on-the-fly. ROOT expands these notebooks by offering a C++ backend in addition to the default Python backend. This allows us to provide a web interface to easily and quickly get started with BioDynaMo. Users do not need to install any software packages; a recent web browser is enough. It is also a convenient tool to interactively go through a demo or tutorial, which opens up possibilities to use BioDynaMo for educational purposes. BioDynaMo is the first biological simulation platform written in C++ that offers such an interface. BioDynaMo notebooks have already been successfully used to demonstrate and teach about pyramidal cell's growth and were well-received by high-school students and teachers during CERN's official teachers and students programs. Figure 1 shows an example of how a BioDynaMo notebook looks. This example gives a brief introduction to pyramidal cells and follows up with a step-by-step explanation of how to simulate their growth with BioDynaMo. Interactive visualizations in the browser give users quick feedback about the simulation status. Lastly, tutorials written as BioDynaMo notebooks can be executed as part of our continuous integration pipeline and ensure that documentation stays in sync with the codebase.



Pyramidal Cell Demo

Hello! In this demonstration we will simulate a "pyramidal cell" with the simulation platform BioDynaMo. A pyramidal cell is one of the many types of neurons that can be found in the human brain. Pyramidal cells are excitatory cells mostly found in the cerebral cortex and the hippocampus, and are implicated in cognitive functions. Pyramidal cells look like the one shown in the image in the right (a pyramidal cell from the visual cortex in mouse).

As you can see the cell body is cone shaped, which is characteristic for this type of neuron, which is why it earned the name "pyramidal cell". These neurons are typically about 100 micrometer in length, but some of the can grow up to a couple of centimeters.

Let's take a look at how we can simulate the development of a pyramidall cell.

```
In [3]:  auto set_param = [&](Param* param) {
           param->bound_space = true;
           param->min_bound = -150;
           param->max_bound = 450;
           double max_length = 2.0;
           param->Get<neuroscience::Param>()->neurite_max_length = 2.0;
           param->cache_neighbors = true;
         };
         bdm::neuroscience::InitModule();
```

Now we can initialize the simulation with an appropriate name and the parameters that we just set.

```
In [4]:  Simulation simulation("pyramidal_cell", set_param);
In [26]: VisualizeInNotebook(800, 400, "y;");
```

Fig. 1. BioDynaMo notebook. A convenient web interface to create and run simulations in a step-by-step manner. The inlining of text and media makes it possible to provide extra information. A few intermediary blocks have been removed to fit the final simulation output on the screenshot.

1.5 Backup and restore

BioDynaMo uses ROOT to integrate the backup and restore functionality transparently. This allows system failures to occur without losing valuable simulation data. Without any user intervention, all simulation data can be persisted to disk as system-independent binary files, called ROOT files, and restored into memory after a failure occurs. The ROOT file format is well-established and is the primary format for storing large quantities (petabytes) of data in high-energy physics experiments, such as CERN. To enable the backup and restore feature in BioDynaMo, one must simply specify the file name of the backup file. Additionally, one can set the interval at which a backup is performed. A low interval value ensures a low amount of data loss whenever a failure occurs, but also increases the incurred overhead for creating the backup files. The advised backup interval depends on the duration of the simulation.

1.6 Code examples

In the main manuscript, Figure 2 depicted in an abstract way that BioDynaMo's software design is open for extension. With the four code examples in Listing 1 to 4—taken directly from the presented use cases and benchmarks—we want to emphasize how little code is required to add new functionality.

```
struct SimParam : public ParamGroup {
  BDM_PARAM_GROUP_HEADER(SimParam, 1);
  uint64_t cells_per_dim = 30;
  uint64_t iterations = 100;
};
```

Listing 1: Additional simulation parameters for the cell growth and division benchmark.

```
/// Possible states.
enum State { kSusceptible, kInfected, kRecovered };

class Person : public Cell {
  BDM_AGENT_HEADER(Person, Cell, 1);

 public:
  Person() {}
  explicit Person(const Double3& position) : Base(position) {}
  virtual ~Person() {}

  /// This attribute stores the current state of the person.
  int state_ = State::kSusceptible;
};
```

Listing 2: New agent class used in the epidemiology use case.

2 Results

2.1 Use cases

The behavior governing apical and basal dendrite growth (neuroscience use case) is outlined in Algorithm 1. Algorithm 2 shows pseudocode for the tumor cell behavior as used in the oncology use case. The epidemiology use case adds three new behaviors which are depicted in Algorithm 3 (infection), Algorithm 4 (recovery), and Algorithm 5 (random movement).

Model parameters can be found in Table 1 for the neuroscience use case, Table 2 for the oncology use case, and Table 3 for the epidemiology use case.

```cpp
struct Recovery : public Behavior {
  BDM_BEHAVIOR_HEADER(Recovery, Behavior, 1);

  Recovery() {}

  void Run(Agent* a) override {
    auto* person = bdm_static_cast<Person*>(a);
    if (person->state_ == kInfected) {
      auto* sim = Simulation::GetActive();
      auto* random = sim->GetRandom();
      auto* sparam = sim->GetParam()->Get<SimParam>();
      if (random->Uniform(0, 1) <= sparam->recovery_probability) {
        person->state_ = State::kRecovered;
      }
    }
  }
};
```

Listing 3: New behavior used in the epidemiology use case.

```cpp
struct TumorDiameterOp : public StandaloneOperationImpl {
  BDM_OP_HEADER(TumorDiameterOp);

  ResultData* result;

  void operator()() override {
    auto* sim = Simulation::GetActive();
    auto* scheduler = sim->GetScheduler();
    auto* param = sim->GetParam();
    auto* sparam = sim->GetParam()->Get<SimParam>();

    // tumor diameter calculation ommitted
    double tumor_diamater = ...
    result->diameter_.push_back(tumor_diameter);

    auto current_time =
        scheduler->GetSimulatedSteps() * param->simulation_time_step_;
    result->time_.push_back(current_time + sparam->starting_iteration);
  }
};
```

Listing 4: New operation to determine the tumor spheroid diameter over time (oncology use case).

Algorithm 1: Apical and basal dendrite growth.

```
   input: neurite, growth_factor, diameter_threshold,
          diameter_threshold_two, growth_speed,
          branching_probability, old_direction_weight,
          randomness_weight, gradient_weight, shrinkage
1  diameter ← neurite.GetDiameter();
2  if diameter > diameter_threshold then
3      old_direction ← neurite.GetDirection();
4      pos ← neurite.GetPosition();
5      gradient ←
        growth_factor.GetNormalizedGradient(pos);
6      direction ← old_direction × old_direction_weight +
        gradient × gradient_weight + RandomUniform3(-1, 1) ×
        randomness_weight;
7      neurite.Extend(growth_speed, direction);
8      neurite.SetDiameter(diameter- shrinkage);
9      if neurite.IsApical() then
10         if neurite.CanBranch() and neurite.IsTerminal()
            and diameter < diameter_threshold_two and
            RandomUniform(0, 1) < branching_probability then
11             branching_direction ←
                CalculateBranchingDirection(neurite);
12             neurite.Branch(branching_direction);
13         end
14     end
15     else if RandomUniform(0, 1) < branching_probability
        then
16         neurite.Bifurcate();
17     end
18 end
```

Table 1. **Model parameters for the pyramidal cell growth simulation.**

Parameter	**Apical dendrite**	**Basal dendrite**
Diameter threshold	0.575	0.75
Diameter threshold two	0.55	
Old direction weight	4	6
Gradient weight	0.06	0.03
Randomness weight	0.3	0.4
Growth speed	100	50
Shrinkage	0.00071	0.00085
Branching probability	0.038	0.006

2.2 Performance

In this section, we provide additional information about our performance evaluation. Table 4 details the experimental environment that we used for our benchmarks and Table 5 lists the measured runtime and memory consumptions on these systems.

Cell growth and division benchmark. The starting condition of this simulation was a 3D grid of cells. These cells were programmed to grow to a specific diameter and divide afterward. This simulation had high cell density and slow-moving cells. This simulation covered mechanical interaction between spherical cells, biological behavior, and cell division.

Soma clustering benchmark. The goal of this model was to cluster two types of cells that are initially randomly distributed. These cells are represented in red and blue in Figure 2A and B. Each cell type secreted a specific extracellular substance which attracted homotypic cells. Substances diffused through the extracellular matrix following Eq 3. We modeled cell processes with two behaviors, ran in sequence: substance secretion (Algorithm 6) and chemotaxis (Algorithm 7). We set the parameter `secretion_quantity` to 1 and `gradient_weight` to 0.75. During the simulation, cell clusters formed depending on their type. The final simulation state after 6000 time steps is shown in Figure 2B. Clusters were associated with non-homogeneous extracellular substance distributions, as shown in Figure 2C and Figure 3. Besides being used as a benchmark, this example demonstrates the applicability of BioDynaMo for modeling biological systems, including the dynamics of chemicals such as oxygen or growth factors. The simulation consisted of 68 lines of C++ code. Table 5 shows the performance on different systems. There are three main differences comparing this simulation with the previous cell growth and division simulation. First, this simulation covered extracellular diffusion. Second, cells moved more rapidly. Third, the number of cells remained constant during the simulation.

Pyramidal cell growth benchmark. We used the pyramidal cell model from the neuroscience use case as a building block (see Figure 3 in the main manuscript). The simulation started with a 2D grid of initial neurons on the z-plane and started growing them. This simulation has three distinctive features. First, activity was limited to a neurite growth front, while the rest of the simulation remained static. This introduced a load imbalance for parallel execution. Second, the neurite implementation modified neighboring agents. Hence, synchronization was required between multiple threads to ensure correctness. Third, the

Algorithm 2: Cancer cell behavior.

```
input: cell, minimum_cell_age, death_probability,
       displacement_rate, growth_speed, division_probability
random_vector ← RandomUniform3(-1, 1);
brownian ← random_vector ÷ random_vector.L2Norm();
cell.UpdatePosition(brownian × displacement_rate);
if age >= minimum_cell_age and
  RandomUniform(0, 1) < death_probability then
    cell.RemoveFromSimulation();
    return;
end
age ← age + 1;
if cell.GetDiameter < max_diameter then
    cell.IncreaseVolume(growth_speed);
else if RandomUniform(0, 1) < division_probability then
    cell.Divide();
end
```

Algorithm 3: Infection behavior.

```
input: person, environment, infection_probability,
       infection_radius
if person.GetState() == susceptible and
  RandomUniform(0,1) < infection_probability then
    neighbors
     ← environment.GetNeighbors(infection_radius);
    for each neighbor in neighbors do
        if neighbor.GetState() == infected then
            person.SetState(infected);
        end
    end
end
```

Algorithm 4: Recovery behavior.

```
input: person, recovery_probability
if person.GetState() == infected and
  RandomUniform(0,1) < recovery_probability then
    person.SetState(recovered);
end
```

Algorithm 5: Random movement behavior.

```
input: person, speed, max_bound
position ← person.GetPosition();
movement ← RandomUniform3(-1, 1).L2Norm();
new_position ← position + movement × speed;
for each el in new_position do
    el ← fMod(el, max_bound);
    if el < 0 then
        el ← max_bound + el;
    end
end
person.SetPosition(new_position);
```

simulation had only static substances, i.e., substance concentrations and gradients did not change over time.

Algorithm 6: Soma clustering substance secretion.

```
input: cell, diffusion_grid, secretion_quantity
pos ← cell.GetPosition();
diffusion_grid.IncreaseConcentrationBy(pos,
  secretion_quantity);
```

Algorithm 7: Soma clustering chemotaxis.

```
input: cell, diffusion_grid, gradient_weight
pos ← cell.GetPosition();
grad ← diffusion_grid.GetNormalizedGradient(pos);
cell.UpdatePosition(grad ×gradient_weight);
```

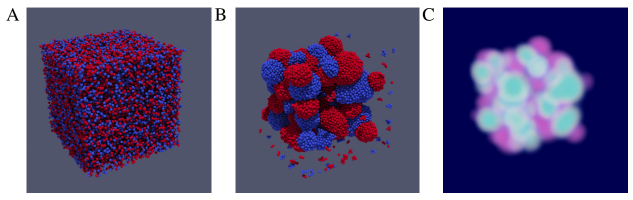


Fig. 2. Soma clustering simulation. This simulation contains two types of cells and two extracellular substances. Each cell secretes a substance and moves into the direction of the substance gradient. Cells are distributed randomly in the beginning (A) and form clusters during the simulation. (B) Cell clusters at the end of the simulation. (C) Substance concentrations at the end of the simulation. A video is available at SV4-soma-clustering.mp4.

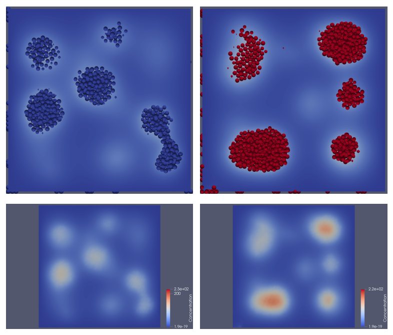


Fig. 3. Soma clustering cross section. Cell positions coincide with regions of high substance concentration. The first row shows substance concentrations and cells, while the second row shows substance concentrations only. Columns show cell type with the corresponding substance.

3 Availability and future directions

BioDynaMo is an open-source project under the Apache 2.0 license and can be found on Github (https://github.com/BioDynaMo/biodynamo). The documentation is split into three parts: API reference, user guide, and developer guide. Furthermore, a Slack channel is available for requesting assistance or guidance from the BioDynaMo development team.

BioDynaMo officially supports the following operating systems: Ubuntu (18.04, 20.04), CentOS 7, and macOS (10.15, 11.00). We test BioDynaMo on these systems and provide prebuilt binaries for third party dependencies: ROOT and ParaView.

All of the results presented in the paper can be reproduced following the instructions in Supplementary Information.

By designing BioDynaMo in a modular and extensible way, we laid the foundation to create new functionalities easily. We encourage the life science community to contribute their developments back to the open-source codebase of BioDynaMo. Over time, the accumulation of all these

Table 2. **Model parameters for the tumor spheroid growth simulations.**

Parameter [dimensions]	2000 cells/well	4000 cells/well	8000 cells/well
Cell growth rate [μm^3/h]	42.0	35.0	29.9
Minimum cell age to apoptosis [h]	87	87	87
Division probability	0.0215	0.0215	0.0215
Cell death probability	0.033	0.033	0.033
Maximum cell speed [μm/h]	1.0	0.9	0.2
Cell–ECM adherence coefficient [dimensionless]	1.8	1.8	1.8
Random cell movement - displacement rate [μm/h]	0.005	0.005	0.0005

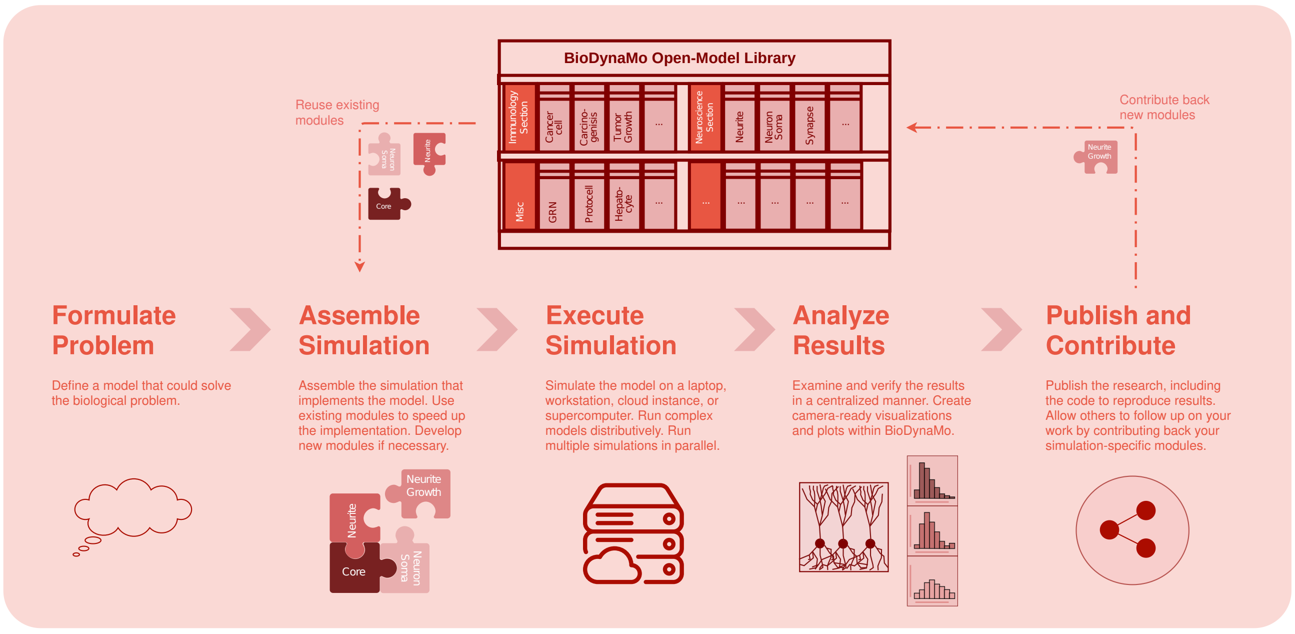

BioDynaMo Open-Model Library
Immunology Section
Cancer cell
Carcino-genisis
Tumor Growth
...
Neuroscience Section
Neurite
Neuron Soma
Synapse
Misc
GRN
Protocell
Hepato-cyte
Reuse existing modules
Contribute back new modules
Neurite Growth
Core
Formulate Problem
Define a model that could solve the biological problem.
Assemble Simulation
Assemble the simulation that implements the model. Use existing modules to speed up the implementation. Develop new modules if necessary.
Execute Simulation
Simulate the model on a laptop, workstation, cloud instance, or supercomputer. Run complex models distributively. Run multiple simulations in parallel.
Analyze Results
Examine and verify the results in a centralized manner. Create camera-ready visualizations and plots within BioDynaMo.
Publish and Contribute
Publish the research, including the code to reproduce results. Allow others to follow up on your work by contributing back your simulation-specific modules.

Fig. 4. BioDynaMo platform. Vision of BioDynaMo, a platform to accelerate in silico experiments.

Table 3. **Model parameters for the epidemiological use case.**

Parameter [dimension]	Measles	Seasonal Influenza
β (analytical solution)	0.06719	0.01321
γ (analytical solution)	0.00521	0.01016
Time step interval [h]	1	1
Number of time steps	1000	2500
Cubic simulation space length [m]	100	100
Initial number of susceptible persons	2000	2000
Initial number of infected persons	20	10
Infection radius [m]	10.50922	10.30043
Infection probability	0.28510	0.05277
Recovery probability	0.00521	0.01016
Max movement per time step [m]	5.78594	4.46516

contributions will form the BioDynaMo open-model library, as shown in Figure 4. This library will help scientists accelerate their research by providing the required building blocks (agents, biological behavior, etc.) for their simulation. Currently, we collect these contributions in our Github repository (https://github.com/BioDynaMo/biodynamo).

4 Supporting information

SF2-reproduce-results.md **Instructions on how to reproduce all results presented in this paper.**

SF3-code.tar.gz **Codebase to reproduce all results presented in this paper.** This file contains all code necessary to reproduce performance results, plots, visualizations, and videos shown in this paper. Furthermore, it contains more details about the hardware and software configuration of the different systems described in Table 4.

SF4-bdm-publication-image.tar.gz **Docker image to reproduce all results presented in this paper.** We provide a Docker image to simplify the process of executing our simulations and benchmarks and to guarantee long-term reproducibility. The only requirement that users must install is a recent version of the Docker engine. All other prerequisites are already provided in the ready-to-use, self-contained Docker image. This approach does not rely on content hosted somewhere on the internet that might become unavailable in the future.

SF5-raw-results.tar.gz **Raw results.** This archive contains all raw results from the simulations and benchmarks shown in this paper.

SV1-single-pyramidal-cell.mp4 **Single pyramidal cell growth simulation, as shown in Figure 3A in the main manuscript.**

SV2-large-scale-neuronal-development.mp4 **Large-scale pyramidal cell growth simulation, as shown in Figure 3C in the main manuscript.**

SV3-tumor-spheroid.mp4 **Tumor spheroid growth simulation, as shown in Figure 4B in the main manuscript.**

SV4-soma-clustering.mp4. **Soma clustering simulation, as shown in Figure 2.**

References

Ahrens, J., Geveci, B., and Law, C. (2005). ParaView: An End-User Tool for Large-Data Visualization. In *Visualization Handbook*, pages 717–731. Elsevier.

Beck, K. and Gamma, E. (2000). *Extreme programming explained: embrace change*. addison-wesley professional.

Bosch, P. S., Ziukaite, R., Alexandre, C., Basler, K., and Vincent, J.-P. (2017). Dpp controls growth and patterning in Drosophila wing precursors through distinct modes of action. *eLife*, **6**, e22546.

Gurdon, J. B. and Bourillot, P.-Y. (2001). Morphogen gradient interpretation. *Nature*, **413**(6858), 797–803.

Kagstrom, S. (2020). SimonKagstrom/kcov. original-date: 2010-08-23T12:03:31Z.

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B. E., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J. B., Grout, J., Corlay, S., *et al.* (2016). Jupyter notebooks-a publishing format for reproducible computational workflows. In *ELPUB*, pages 87–90.

Miller, G. (2006). A Scientist's Nightmare: Software Problem Leads to Five Retractions. *Science*, **314**(5807), 1856–1857.

Pattana, S. (2006). *Division d'un milieu cellulaire sous contraintes mécaniques: utilisation de la mécanique des matériaux granulaires*. Ph.D. thesis, Université Montpellier II, place Eugéne Bataillon 34095 Montpellier Cedex 5 France.

Smith, G., Smith, G., Smith, G., Smither, M., and Press, O. U. (1985). *Numerical Solution of Partial Differential Equations: Finite Difference Methods*. Oxford applied mathematics and computing science series. Clarendon Press.

Zubler, F. and Douglas, R. (2009). A framework for modeling the growth and development of neurons and networks. *Frontiers in computational neuroscience*, **3**, 25.

Table 4. **Experimental environment.** Main parameters of the systems that we used to run the benchmarks of this paper. SF3-code.tar.gz contains more details.

System	Main memory	CPU / GPU	OS
A	504 GB	Server with four Intel(R) Xeon(R) E7-8890 v3 CPUs @ 2.50GHz with a total of 72 physical cores, two threads per core and four NUMA nodes.	CentOS 7.9.2009
B	1008 GB		
C	191 GB	Server with two Intel(R) Xeon(R) Gold 6130 CPUs @ 2.10GHz with 16 physical cores, two threads per core, and two NUMA nodes. One NVidia Tesla V100 SXM2 GPU with 32 GB memory.	CentOS 7.7.1908
D	16 GB	Dell Latitude 7480 Laptop from 2017. One Intel(R) Core(TM) i7-7600U CPU @ 2.80GHz with two physical cores and two threads per core. One Intel HD Graphics 620 GPU with 64 MB eDRAM.	Ubuntu 20.04.1 LTS

Table 5. **Performance data.** The values in column “Agents" and “Diffusion volumes" are taken from the end of the simulation. Runtime measures the wall-clock time to simulate the number of iterations. It excludes the time for simulation setup and visualization.

Simulation	Agents	Diffusion volumes	Iterations	System (Table 4)	Physical CPUs	Runtime	Memory
Neuroscience use case							
Single (Figure 3A in the main manuscript)	1 494	250	500	A	1	0.16 s	375 MB
				D	1	0.13 s	471 MB
Large-scale (Figure 3C in the main manuscript)	9 041 632	65 536	500	A	72	45 s	6 GB
				D	2	10 min 53 s	5.3 GB
Very-large-scale	1 018 280 997	5 606 442	500	B	72	1 h 37 min	507 GB
Oncology use case (Figure 4 in the main manuscript)							
2000 initial cells	4 169	0	312	A	1	1.24 s	378 MB
				D	1	0.98 s	475 MB
4000 initial cells	5 241	0	312	A	1	2.09 s	380 MB
				D	1	1.62 s	477 MB
8000 initial cells	8 225	0	288	A	1	4.16 s	381 MB
				D	1	3.39 s	477 MB
Large-scale	10 123 903	0	288	A	72	2 min 20 s	9.02 GB
				D	2	53 min 31 s	5.94 MB
Very-large-scale	1 012 302 977	0	288	B	72	6 h 49 min	639 GB
Epidemiology use case (Figure 5C in the main manuscript)							
Measles	2 010	0	1000	A	1	0.59 s	372 MB
Seasonal Influenza	2 020	0	2500	A	1	1.59 s	373 MB
Large-scale (measles)	10 050 000	0	1000	A	72	1 min 25 s	6.02 GB
				D	2	54 min 22 s	3.61 GB
Very-large-scale (measles)	1 005 000 000	0	1000	B	72	3 h 54 min	326 GB
Soma clustering (Figure 2)	32 000	1 240 000	6 000	A	72	12.27 s	1.26 GB
				D	2	2 min 20 s	515 MB