BioDynaMo: an agent-based simulation platform for scalable computational biology research

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

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 Biosciences Institute, Newcastle University, Newcastle upon Tyne, United Kingdom
5 Department of Mechanical & Manufacturing Engineering, University of Cyprus, Nicosia, Cyprus
6 Department of Medical Physics & Biomedical Engineering, University College London, London, UK
7 School of Computing, Newcastle University, Newcastle upon Tyne, UK
8 Department of Functional Neurosurgery, Ruijin Hospital, Shanghai Jiao Tong University School of Medicine, Shanghai, China
9 SCimPulse Foundation, Geleen, Netherlands

* lukas.breitwieser@inf.ethz.ch, a.s.hesam@tudelft.nl, roman.bauer@newcastle.ac.uk

Keywords

simulation, agent-based, biological dynamics, cell proliferation, parallel computing, high-performance computing, neuroscience, neural development, C++, open-source

Abstract

Computer simulation is an indispensable tool for studying complex biological systems. In particular, agent-based modeling is an attractive method to describe biophysical dynamics. However, two barriers limit faster progress. First, simulators do not always take full advantage of parallel and heterogeneous hardware. Second, many agent-based simulators are written with a specific research problem in mind and lack a flexible software design. Consequently, researchers have to spend an unnecessarily long time implementing their simulation and have to compromise either on model resolution or system size.

We present a novel simulation platform called BioDynaMo that alleviates both of these problems researchers face in computer simulation of complex biological systems. BioDynaMo features a general-purpose and high-performance simulation engine. The engine simulates cellular elements, their interactions within a 3D physical environment, and their cell-internal genetic dynamics. Cell-internal dynamics can be described in C++ code or using system biology markup language (SBML).

We demonstrate BioDynaMo's wide range of application with three example use cases: soma clustering, neural development, and tumor spheroid growth. We validate our results with experimental data, and evaluate the performance of the simulation

engine. We compare BioDynaMo's performance with a state-of-the-art baseline, and analyze its scalability. We observe a speedup of 20–124× over the state-of-the-art baseline using one CPU core and a parallel speedup between 67× and 76× using 72 physical CPU cores with hyperthreading enabled. Combining these two results, we conclude that, on our test system, BioDynaMo is at least three orders of magnitude faster than the state-of-the-art serial baseline. These improvements make it feasible to simulate neural development with 1.24 billion agents on a single server with 1TB memory, and 12 million agents on a laptop with 16GB memory.

BioDynaMo is an open-source project under the Apache 2.0 license and is available at www.biodynamo.org.

Author summary

Computer simulations of biological systems are crucial to gain insights into complex processes of living organisms. However, the development of increasingly large and complex simulations is a difficult task, partly because a strong background in biology as well as computer science is required. In this paper, we introduce BioDynaMo, an agent-based simulation platform with which life scientists can create simulations that are three orders of magnitude faster than the state-of-the-art baseline. By taking advantage of the latest developments in computing hardware, we build a platform that is highly optimized. This enables the simulation of 1.24 billion agents on a single server and 12 million agents on a laptop. BioDynaMo places a lot of focus on hiding computational complexity and providing an easy-to-use interface, such that the life scientist can concentrate on biological aspects, rather than computational. BioDynaMo helps scientists to translate an idea quickly into a simulation by providing common building blocks, and a modular and extensible software design. We analyze the performance of the platform and demonstrate the capabilities with three example use cases: soma clustering, neural development, and tumor spheroid growth. The results support the view that BioDynaMo will help open up new research opportunities for large-scale biological simulations.

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 [1, 2].

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 [3] and Dennard scaling [4], 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. 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 have developed a new platform called 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 [5], 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, neurite growth, etc.

BioDynaMo provides by design seven system properties:

- **Agent-based.** The BioDynaMo project was established to support developmental simulations of biological dynamics. A good approximation for such simulations is agent-based modeling [6]. Cells, or agents, are modeled as discrete objects that perform actions based on their current cell state, "genetic makeup", and the surrounding microenvironment. Similar to nature, where a single cell, the zygote, develops into a whole organism, a BioDynaMo simulation can be started with a single cell that contains all the genetic rules necessary for its differentiation, proliferation, and cell function.

- **General purpose.** BioDynaMo is developed to become a general-purpose tool for agent-based biological simulation. To simulate models from various fields of biology, BioDynaMo's software design is extensible and modular.

- **Multi scale.** 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.). Hence, the system is designed to support models that comprise different time scales.

- **Large scale.** Biological systems contain a large number of cells. The cerebral cortex, for example, comprises approximately 16 billion neurons [7]. Biologists should not be limited by the number of cells 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 main requirements BioDynaMo satisfies.

  First, BioDynaMo provides a wide range of common functionality like 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.

- **Integrated.** BioDynaMo integrates with well-established open standards in the domain of computational biology to enable reuse of prior models. Its interoperability with other software ecosystems is likely to increase the productivity of researchers by reducing programming effort.

  An example of such a standard that BioDynaMo integrates well with is the systems biology markup language (SBML) [8]. There exists a large number of

published chemical reaction networks in this format together with a mature software collection that includes solvers, editors, viewers, and utilities [9].

This paper makes the following contributions: (i) We present a novel high-performance and general-purpose simulation platform for agent-based simulations. (ii) We detail the user-facing features of BioDynaMo that enable users to build a simulation based on predefined building blocks and to define a biological model tailored to their needs. (iii) We present three example simulations to demonstrate BioDynaMo's capabilities and modularity: soma clustering, pyramidal cell growth, and tumor spheroid growth. (iv) We show that BioDynaMo can produce biologically-meaningful simulation results by validating them against experimental data. (v) We present performance data on different systems and scale the pyramidal cell example up to 1.24 billion agents to demonstrate BioDynaMo's performance.

Prior work

The existence and continued use of many agent-based simulators [5, 10–20] demonstrates the importance of agent-based models in computational biology research. In this section, we compare BioDynaMo's most crucial system properties with prior work.

Large-scale model support. The authors of BioCellion [12], PhysiCell [16], and Timothy [17] 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 [15] 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 [10, 11, 14, 18], cell colonies [12, 17, 19], and neural development [5, 13, 15]. Such specialization makes it hard to adapt these simulators to different use cases. In contrast, BioDynaMo, with its modularity and extensibility, provides a general-purpose platform for agent-based biological simulations.

Quality assurance. Automated software tests are the foundation of a modern development workflow. Unfortunately, the authors of several simulation tools [5, 13–15, 17, 18] omit these tests. Mirams et al. [19] recognize this shortcoming and describe a rigorous development workflow in their paper. BioDynaMo uses a similar approach to ensure high code quality and promote reproducibility.

SBML integration. Existing agent-based simulators [5, 10–20] do not integrate with the well-established SBML standard. Therefore, users of these simulators do not benefit from the large collection of available chemical reaction networks described in SBML format. In contrast, BioDynaMo allows users to simulate SBML models in each agent.

Design and implementation

In this section, we present the main simulation concepts of BioDynaMo, describe our approach to achieve modularity and high performance, and outline our development process and principles to ensure high software quality.

Simulation concepts

BioDynaMo supports biological simulations that follow an agent-based approach. Fig 1 gives an overview of BioDynaMo's main concepts. An agent—or `simulation object` in the code—has a 3D geometry, behavior, and microenvironment. 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 (Fig 1C). This behavior is defined by *biology modules*, which can be assigned to agents. Typically, the complete agent behavior is composed of multiple assigned biology modules, each with a specific purpose (e.g. mitosis, substance secretion, chemotaxis, etc.).

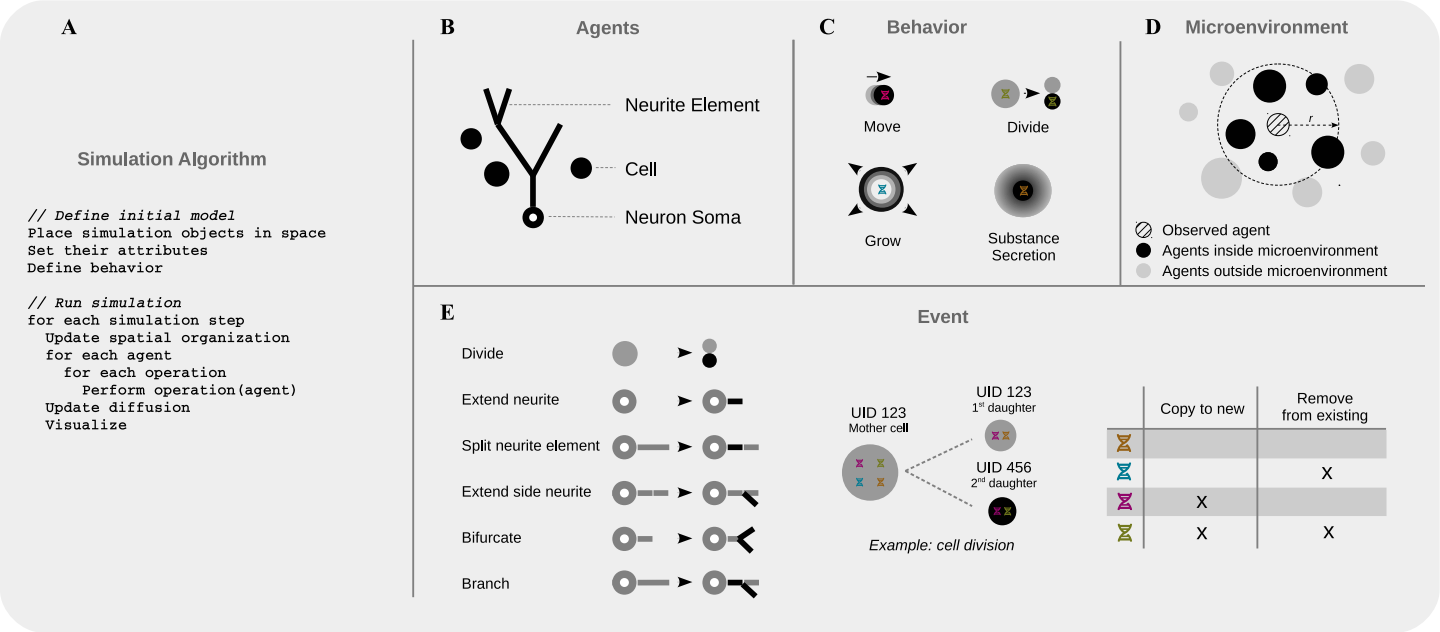

Fig 1. Simulation concepts. Overview of the high-level simulation concepts of BioDynaMo. The simulation algorithm (A) can be divided into two main parts: the definition of the initial model and execution of the simulation. Agents (B) have their own geometry, behavior (C), and microenvironment (D). (C) Agent behavior is defined by biology modules that are inserted into agents. A few possible examples are displayed. The sum of all biology modules of an agent determines the simulated genetic makeup. (E) BioDynaMo's event system provides a uniform framework to handle the creation of new agents. A list of currently supported events is displayed on the left column. Events help to regulate the simulated genes if a new agent is created. They offer a simple way to encode which biology module should be copied to the new agent and which ones should be removed from the existing one. The middle and right image illustrates this mechanism for the cell division event. The update of an agent is based on its current state and its surrounding microenvironment. (D) The microenvironment is determined by radius r and contains other agents or extracellular substances. S1 Table describes agents, events, and operations in more detail.

Currently, BioDynaMo offers three predefined agents: a generic cell, a neuron soma, and a neurite element (Fig 1B). The simulation engine executes operations for all agents for each time step. At a high level, an *operation* is a function that: (i) alters the state of an agent and potentially also its microenvironment, (ii) creates a new agent, or (iii) removes an agent from the simulation. BioDynaMo supports processes at different time scales by providing 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. S1 Table describes the predefined agents, events, and operations of BioDynaMo in more detail. The results section contains information on the biology modules.

The *microenvironment* is the volume that the agent can interact with (Fig 1D). It comprises other agents and chemical substances in the extracellular matrix. Surrounding agents are, for example, needed to calculate mechanical interactions among agent pairs.

Currently, the user defines a simulation programmatically in C++ (Fig 1A). 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. The simulation engine, among other things, calculates mechanical interactions between agents, calculates the diffusion of substances, and executes the behavior of each agent (see operations in S1 Table).

Events. An event denotes the creation of new agents during a simulation. Consider cell division as an example: the mother cell splits into two genetically identical daughter cells (Fig 1E). First, BioDynaMo creates a new cell (second daughter) and initializes its attributes such as the volume. The volume is determined by the volume ratio between the two daughter cells, and the volume of the original mother cell. Second, BioDynaMo transforms the existing mother cell to become the first daughter. To achieve this, the attributes of the mother cell must be modified. For instance, BioDynaMo reduces the volume such that the given volume ratio between the two daughter cells is satisfied, and the sum of the volume of the two daughter cells is equal to the volume of the previous mother cell. BioDynaMo must execute similar instructions for all other data members.

In general, an event is triggered by *one* agent and can create multiple new agents. The biologist must decide how the attributes of the new agents should be initialized and how the attributes of the triggering agent should be modified. Therefore, BioDynaMo offers the event system to provide a uniform method to express these decisions.

The event system consists of three parts: (i) the event class that encapsulates event parameters (e.g. volume ratio), (ii) the event constructor that initializes the data members of the new agents, and (iii) the event handler function that modifies data members of the existing agent.

Every event class inherits from class `Event`. `CellDivisionEvent`, for example, contains two data members (i.e. attributes): the volume ratio between the two daughter cells, and the division axis. The event constructor of the agent takes two arguments: the event object and a pointer to the triggering agent (the mother cell in the cell division example). Consequently, all required information is available in the function body of the constructor to initialize the data members of the new agent. Similarly, the event handler of the agent takes the event object and a pointer to each newly created agent as arguments.

BioDynaMo's event system provides a uniform framework to handle the creation of new agents and makes it simple to extend their classes. If users, for example, extend BioDynaMo's `Cell` implementation by adding a new data member, they can define how the new data member should be modified, if their custom cell divides. To achieve this, they only have to define an event constructor and event handler function for their new class.

Behavior. The virtual genetic makeup responsible for the behavior of an agent and its phenotype is encoded as a set of biology modules (Fig 1C). Example biology modules are substance secretion, chemotaxis, or cell division. The main component of a biology module is its `Run` function, which contains the instructions that are executed at every

time step. Inside this function, the user has access to the corresponding agent and its microenvironment.

Like genes, biology modules can be activated or inhibited. BioDynaMo achieves this by attaching them to or removing them from the corresponding agent. BioDynaMo simplifies the regulation of biology modules for events. The user can control if a biology module will be copied to a new agent or removed from the triggering agent by providing two event id lists. All possible combinations of "copy to new" and "remove from existing" are illustrated in Fig 1E. for the cell division event.

Microenvironment. BioDynaMo determines the microenvironment (Fig 1D) 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 microenvironment 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.

Mechanical interaction. 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 microenvironement. 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 [5]. Both in BioDynaMo and Cortex3D, the magnitude of the force is computed based on [21] 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 and/or cylinder.

Eq 1 comprises the effects due to 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 [5].

Extracellular diffusion. Passive transport [22] is an important concept in developmental biology. Signaling molecules, which differentiate and regulate cells, reach their destination through diffusion [23]. A well-studied example of this process, called morphogen gradients, is the determination of vein positions in the wing of Drosophila [24].

BioDynaMo solves the partial differential equations that model the diffusion of extracellular substances (Fick's second law) with the discrete central difference

scheme [25]. 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 (see results section).

Modularity

BioDynaMo aims to be a simulation framework that can be used to develop simulations in various different life science fields (e.g. immunology, neuroscience, 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. Many of these support features are based on the ROOT framework [26] that has been developed at CERN to process the experimental data from the large hadron collider. Functionalities that are field-specific are separated from the core and are bundled as a separate module.

Fig 2 gives an overview of the layered architecture of BioDynaMo. As mentioned earlier, BioDynaMo core provides basic functionality and hides implementation details from lower layers. A biologist can build a whole simulation exclusively using functionality from BioDynaMo core. If needed, further functionality can be included by using BioDynaMo modules or third party libraries. BioDynaMo can even be coupled with other simulators e.g., to create a hybrid agent-based, continuum-based model [27].

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 [5]. 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 [5].

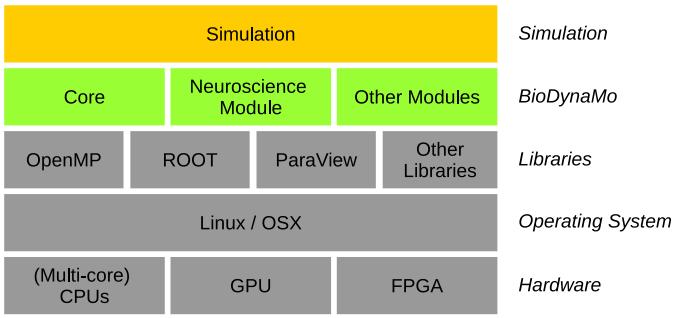

Fig 2. BioDynaMo abstraction layers. Functionality is divided between BioDynaMo core and optional BioDynaMo modules. BioDynaMo core hides implementation details from lower layers, like parallelization, visualization, or simulation backups, while offering a rich set of simulation primitives. Every simulation is based on BioDynaMo core. Furthermore, the biologist can include functionality from different BioDynaMo modules, and third-party libraries that are not part of BioDynaMo.

User-defined components. If the desired functionality is missing, the user can create, extend, or modify agents, biology modules, events, and operations in the "Simulation" layer of Fig 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, which ultimately helps researchers to implement their models more rapidly.

Performance and parallelism

We parallelized the whole simulation engine using OpenMP [28] compiler directives. OpenMP is a good fit since BioDynaMo exploits mostly loop parallelism (see Fig 1A). Our GPU code is implemented in CUDA [29] and OpenCL [30] and can be executed on graphics cards of different vendors (NVidia, AMD, or Intel). Offloading computation to a GPU can improve performance, especially on laptops were the number of CPU cores is limited.

To increase the maximum theoretical speedup due to parallel processing (as described by Ahmdahl's law [31]), we minimize the number of serial code portions in BioDynaMo. 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 microenvironment 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 built-in support for such functionality. We aim to provide more details on BioDynaMo's performance enhancements and analyses in a future paper targeting computer scientists.

Visualization

BioDynaMo currently uses ParaView [32] 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 Fig 5), 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 Fig 5, for example, to color cells by their cell type.

SBML integration

Systems biology markup language (SBML) is a well-established standard to describe chemical reaction networks to model metabolism, or cell signaling [8]. The BioModels database contains a collection of more than 9000 models in this format [33].

Therefore, BioDynaMo provides the possibility to simulate chemical reaction networks described as SBML [8] models (Fig 3). BioDynaMo uses libroadrunner to solve the reaction equations, which features various deterministic and stochastic solvers. The intracellular concentration of substances can serve as a control mechanism for agent behaviors displacement, division, branching, etc. Furthermore, it is feasible to couple the intracellular domain with the extracellular matrix by exocytosis and endocytosis of substances and extracellular diffusion.

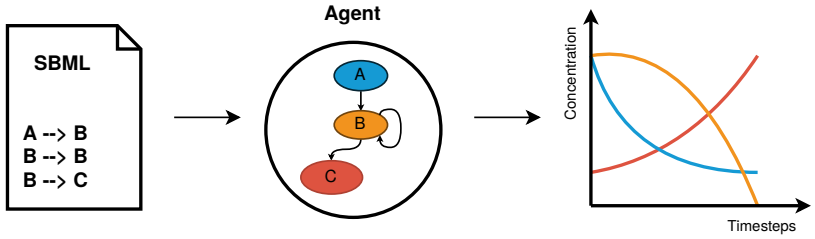

Fig 3. SBML integration. Chemical reaction networks defined in SBML format can be loaded into BioDynaMo and assigned to any agent. The reaction equations are solved for each timestep.

Software quality assurance

Compromising on software quality can have severe consequences that can culminate in the retraction of published manuscripts [34]. Therefore, we put tremendous effort into establishing a rigorous development workflow that follows industry best practices. Test-driven development—a practice from agile development [35]—is at the core of our solution. BioDynaMo has more than 200 tests distributed among unit, convergence, system, and installation tests. Fig 4, for example, shows the convergence test for extracellular diffusion. By comparing the computed results with the analytical solution we observe an increasing accuracy of the diffusion solver when we increase the diffusion grid's resolution. For each change to our repository [36], Travis-CI [37] executes the entire test suite and, upon success, updates the documentation on our website [38]. Installation tests are executed on each supported operating system and ensure that all demo simulations run on a default system.

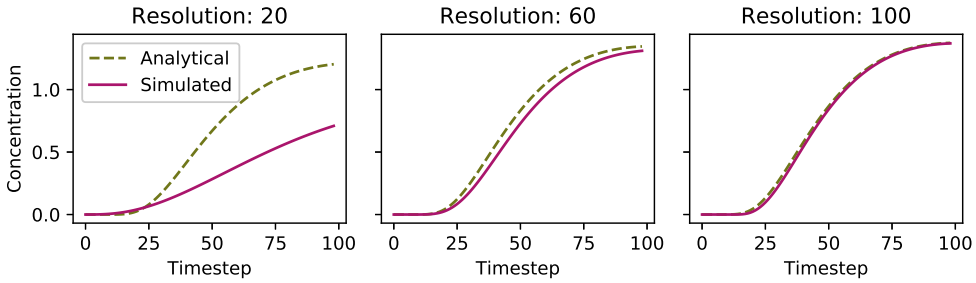

Fig 4. Diffusion convergence test. The simulated diffusion results converge towards the analytical solution as we increase the resolution of the diffusion grid. The resolution represents the number of grid points in the diffusion grid along each dimension of the simulation space. We use an instantaneous point source at the origin and measure the concentration change over time at $\sqrt{1000}$ micron away from the point source.

Results

This section provides evidence that our presented design meets the desired system properties. We demonstrate that BioDynaMo can be used to model a wide range of biological processes with three example use cases: soma clustering, neural development, and tumor spheroids. Furthermore, we compare BioDynaMo's performance with a serial neural simulator, analyse its scalability, and quantify the impact of GPU acceleration. Table 1 details the experimental environment used for performance analysis.

Table 1. Experimental environment. Main parameters of the systems that we used to run the benchmarks of this paper. S2 File contains more details.

System	Main memory	CPU / GPU	OS / Compiler
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 7.7.1908 g++ in the version 7.3
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.	
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 16.04.4 LTS g++ in the version 5.5.0

Example application areas

Soma clustering

This example demonstrates soma clustering using BioDynaMo. We modeled the formation of cell aggregations over time in response to external factors. Initially, we created 32,000 randomly distributed cells of two different types in a cubic volume. These cells are represented in red and blue in Fig 5A 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 behavior with two biology modules, ran in sequence: substance secretion (Algorithm 1) and chemotaxis (Algorithm 2). 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 timesteps is shown in Fig 5B. Clusters were associated with non-homogeneous extracellular substance distributions, as shown in Fig 5C. 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 158 lines of code. The runtime on a server with 72 CPU cores was 15.88s and 2min 20s on a laptop with two CPU cores (Table 2).

Algorithm 1: Soma clustering substance secretion.

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

Algorithm 2: Soma clustering chemotaxis.

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

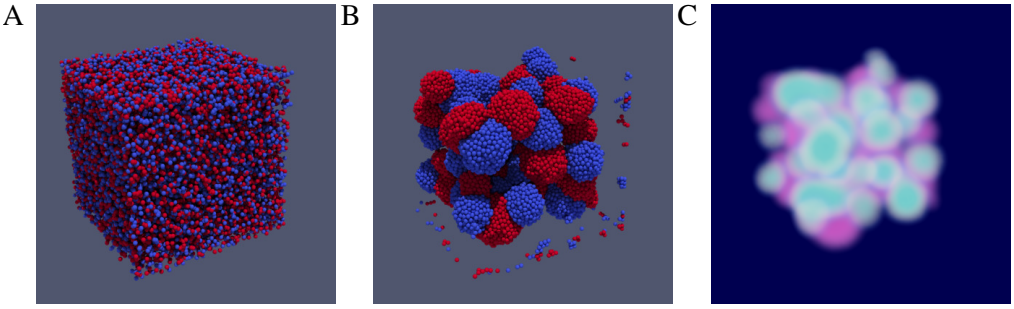

Fig 5. Soma clustering simulation. This simulation contains two types of cells and two extracellular substances. Each cell secretes a substance and moves into the direction of the substance gradient. Cells are distributed randomly in the beginning (A) and form clusters during the simulation. (B) Cell clusters at the end of the simulation. (C) Substance concentrations at the end of the simulation. A video is available at S1 Video.

Neural development

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— 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 biological module defining growth direction, speed, and branching behavior for apical and basal dendrites. At each step of the

Table 2. Performance data for example use cases. 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 1)	Physical CPUs	Runtime	Memory
Soma clustering (Fig 5)	32 000	1 240 000	6 000	A	72	15.88 s	663 MB
				D	2	2 min 20 s	73 MB
Pyramidal cell							
Single (Fig 6A)	2 080	250	500	A	1	0.22 s	32 MB
				D	1	0.14 s	19 MB
Large-scale (Fig 7)	12 336 727	65 536	500	A	72	1 min 13 s	6.96 GB
				D	2	13 min 5 s	6.66 GB
Very-large-scale	1 238 356 383	4 812 208	500	B	72	2 h 14 min	603 GB
Tumor spheroid (Fig 8)							
2000 initial cells	4 169	0	312	A	1	1.22 s	87 MB
				D	1	0.92 s	107 MB
4000 initial cells	5 241	0	312	A	1	2.07 s	87 MB
				D	1	1.53 s	108 MB
8000 initial cells	8 225	0	288	A	1	4.33 s	90 MB
				D	1	3.34 s	111 MB

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 (0.03 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. Algorithm 3 shows pseudocode for apical and basal dendrite growth while S1 File shows chosen values for each parameter.

These simple rules gave rise to a straight long apical dendrite with a simple branching pattern and more dispersed basal dendrites, as shown in Fig 6A, similar to what can be observed in real pyramidal cell morphologies. 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 [39] for comparison and parameter tuning. Two measures were used to compare our simulated neurons and the 107 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 Fig 6B. The simulation of the pyramidal cell growth consisted of 392 lines of code. Simulation runtime on a server with 72 physical cores was 15.88s and 2min 20s on a laptop with

Algorithm 3: 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
diameter ← neurite.GetDiameter();
if diameter > diameter_threshold then
    old_direction ← neurite.GetDirection();
    pos ← neurite.GetPosition();
    gradient ← growth_factor.GetNormalizedGradient(pos);
    direction ← old_direction × old_direction_weight + gradient × gradient_weight +
     RandomUniform3(-1, 1) × randomness_weight;
    neurite.Extend(growth_speed, direction);
    neurite.SetDiameter(diameter- shrinkage);
    if neurite.IsApical() then
        if neurite.CanBranch() and neurite.IsTerminal() and
         diameter < diameter_threshold_two and
         RandomUniform(0, 1) < branching_probability then
            branching_direction ← CalculateBranchingDirection(neurite);
            neurite.Branch(branching_direction);
        end
    end
    else if RandomUniform(0, 1) < branching_probability then
        neurite.Bifurcate();
    end
end
```

two CPU cores (Table 2).

Fig 7 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 more than 12 million agents. This, however, is not the limit: in another simulation, we simulated 1.24 billion agents on a single server. On System B (Table 1), the latter simulation took 2 hours 14 minutes for 500 iterations using all CPU cores. These 500 iterations correspond to approximately three weeks of pyramidal cell growth in the rat.

Tumor spheroid growth

In this section, we present a tumor spheroid simulation to replicate in vitro experiments from [40]. Tumor spheroid experiments are typically employed to investigate the pathophysiology of cancer, and are also being used for pre-clinical drug screening [41]. Here we considered three in vitro test cases using a breast adenocarcinoma MCF-7 cell line [40] 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 microenvironment.

Initially, cells 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 1 mm^3 cube. The fundamental cellular mechanisms modeled here include cell growth, cell duplication, cell migration, and cell apoptosis. A single biology module (Algorithm 4) governed all these processes. The model parameters are listed in S1 File. The cell growth rate was derived from the published data [42],

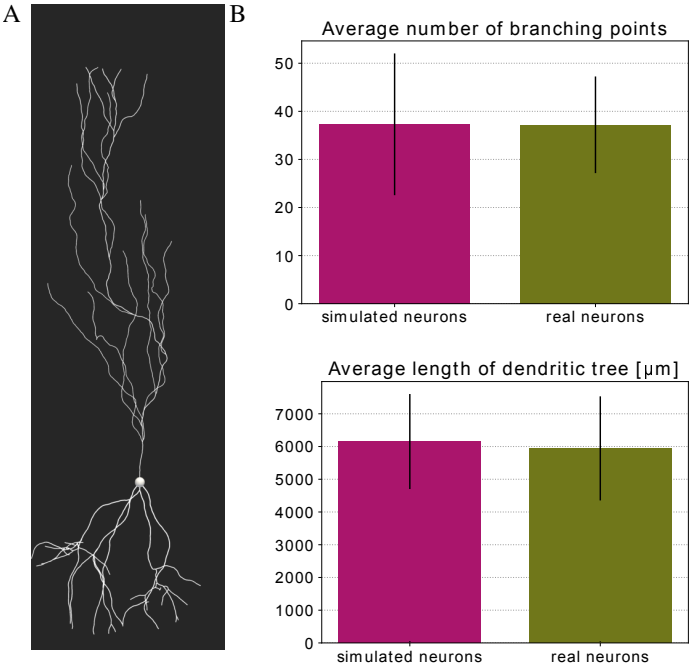

Fig 6. Pyramidal cell simulation. (A) Example pyramidal cell simulated with BioDynaMo. A video is available at S2 Video. (B) Morphology comparison between simulated neurons and experimental data from [39]. Error bars represent the standard deviation.

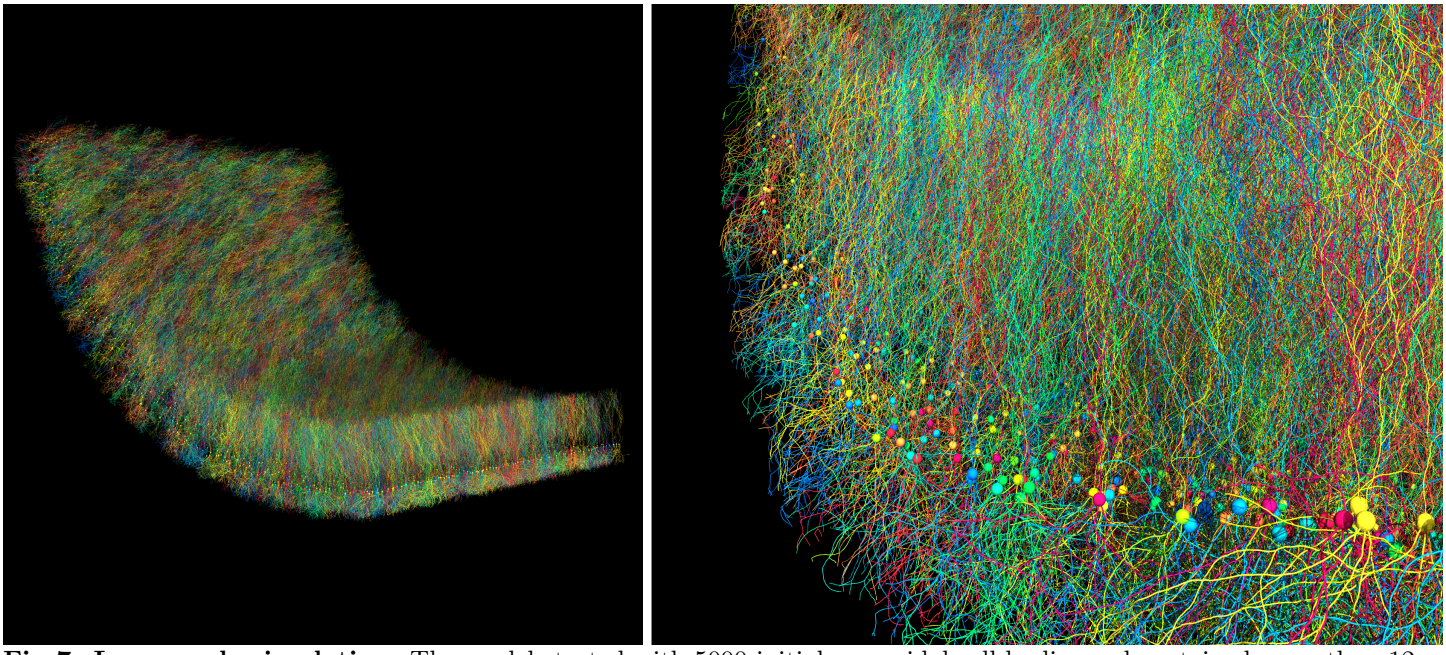

Fig 7. Large-scale simulation. The model started with 5000 initial pyramidal cell bodies and contained more than 12 million agents after simulating 500 iterations. Simulation execution time was 73 seconds on a server with 72 CPU cores. A video is available at S3 Video.

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 present 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.

Algorithm 4: 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
```

The in vitro experiments showed that instantaneous spheroid growth was hindered by the compression of the surrounding agarose gel matrix (see Fig 8A), 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 Fig 8A, BioDynaMo simulations are set to start on day two or three.

The in vitro experiments from [40] and the simulations using BioDynaMo are depicted in Fig 8. Each line plot in Fig 8A 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 [40]. Fig 8B 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 441 lines of code, including the generation of the plot shown in Fig 8A. Running one simulation took 0.92–3.34s on a laptop and 1.22–4.33s on a server, both using one CPU core (see Table 2).

Measuring the performance of BioDynaMo

The efficient usage of available computing resources is crucial to simulate large-scale and detailed biological processes. 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. We first describe these simulations in more detail.

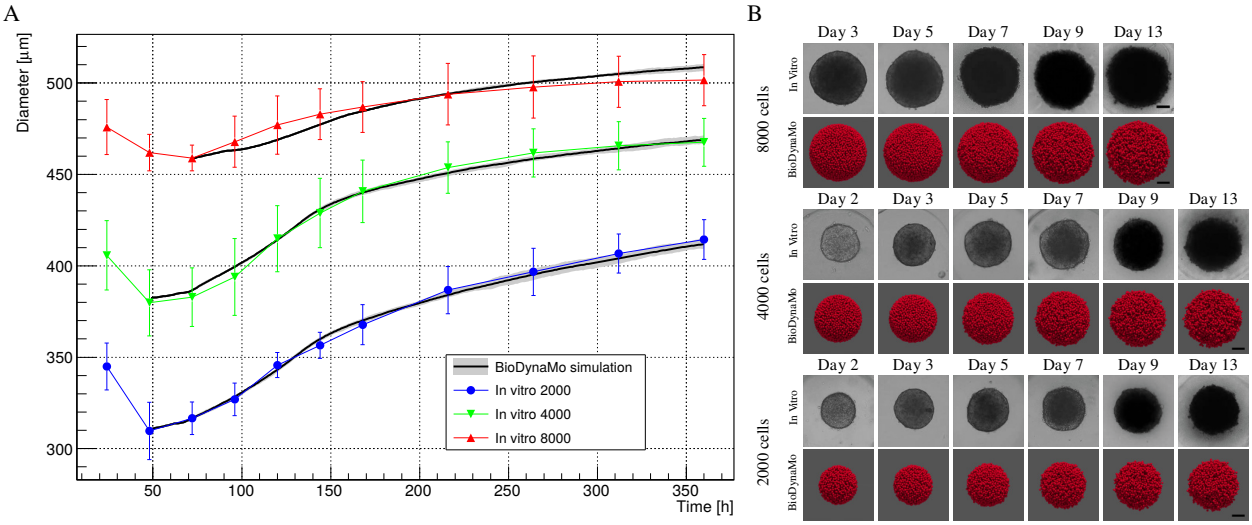

Fig 8. Comparison between in vitro MCF-7 tumor spheroid experiments and our in silico simulations using BioDynaMo. (A) Human breast adenocarcinoma tumor spheroid (MCF-7 cell line) development during a 15 day period, where different initial cell populations were considered (see Fig 3 in [40]). Error bars 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 at S4 Video.

Cell growth and division. 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. The goal of this model was to cluster two types of cells that are initially randomly distributed. We used the same implementation presented in the example use cases (see Fig 5). 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. We used the pyramidal cell model described in the example use cases as a building block (see Fig 6). 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 simulation had only static substances, i.e., substance concentrations and gradients did not change over time.

We evaluated BioDynaMo's performance by comparing it to prior work, analyzing its scalability, and quantifying the performance improvement of GPU acceleration.

First, to demonstrate the performance improvements over prior work, we compared BioDynaMo with the Cortex3D [5] neuroscientific simulator. Cortex3D has the highest

similarity in terms of the underlying biological model out of all the related works presented. This makes Cortex3D the best candidate with which to compare BioDynaMo and ensure a fair comparison. Fig 9A shows the speedup of BioDynaMo for the three simulations. We observed a significant speedup between 20 and 124×. 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.

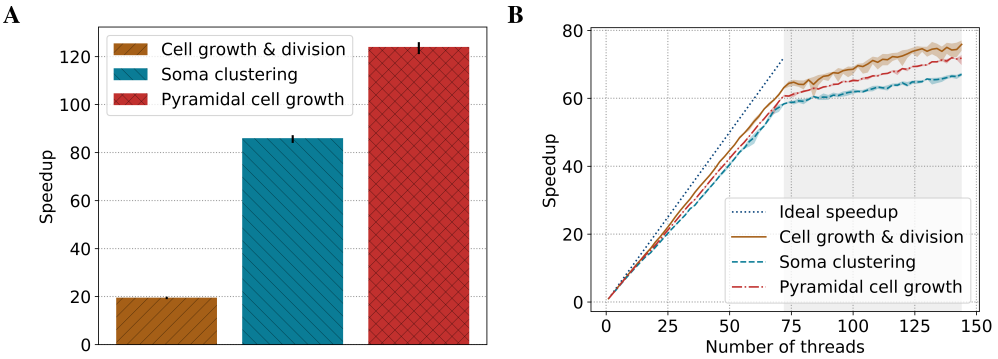

Fig 9. BioDynaMo performance analysis. (A) Speedup of BioDynaMo compared to Cortex3D. (B) Strong scaling behavior of BioDynaMo on a server with 72 physical cores, two threads per core, and four NUMA domains. The grey area highlights hyper-threads.

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. Fig 9B shows the strong scaling analysis. All simulation parameters remained constant, and the number of threads was increased from one to the number of logical cores provided by the benchmark server. The maximum speedup was between 67× and 76×, which corresponds to a parallel efficiency of 0.93 and 1.06. Performance improved even after all physical cores were utilized and hyper-threads were used. Hyper-threads are highlighted in gray in Fig 9B. 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 (Table 1), comparing an NVidia Tesla V100 GPU with one CPU core. We observed a speedup of 1.48× for cell growth and division, and 4.05× 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 67×. Furthermore, we measured a minimum speedup of 20× comparing BioDynaMo with Cortex3D both using a single thread. Based on these two observations, we conclude that on System A (Table 1) BioDynaMo is more than three orders of magnitude faster than Cortex3D.

Discussion

This paper presented BioDynaMo, a novel open-source platform for agent-based simulations. We demonstrated BioDynaMo's modular architecture with three example use cases: soma clustering, pyramidal cell growth, and tumor spheroid growth.

Modularity, predefined simulation primitives, and common functionality make it easy to translate an idea into a BioDynaMo simulation. All examples were implemented in 158 to 441 lines of code.

Our performance analysis on System A (Table 1) showed a three-order-of-magnitude improvement over the state-of-the-art baseline Cortex3D. This improvement enables simulations with an unprecedented number of agents. We simulated neural development with a final number of 1.24 billion agents after 500 iterations in 2 hours and 14 minutes. Simulations can also be run on less powerful hardware. A laptop took slightly over 13 minutes to complete 500 iterations of the pyramidal cell growth simulation with 12 million agents.

We established a rigorous development workflow that helps to reach high software quality and reproducibility of results. In future works, we plan to focus on technical details of the simulation engine, a distributed runtime to support even larger simulations on multiple servers, and improved hardware acceleration.

To the best of our knowledge, BioDynaMo is the first scalable simulator of neuronal growth and interactions between cells. It helps life-scientists harness the computing power of modern hardware and hides the technical details of high-performance parallel computing. Thus, life-scientists can concentrate on gaining a mechanistic understanding of complex biological processes, which can even yield predictions of clinical progression [43, 44].

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

Availability and future directions

BioDynaMo is an open-source project under the Apache 2.0 license and can be found on Github [36]. 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 (16.04, 18.04), CentOS 7, and macOS. We test BioDynaMo on these systems and provide prebuilt binaries for third party dependencies: ROOT, ParaView, and libRoadrunner.

All of the results presented in the paper can be reproduced following the instructions in S2 File.

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 contributions will form the BioDynaMo open-model library, as shown in Fig 10. 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 contrubutions in our Github repository [36].

Supporting information

All supporting information is available at:
`https://doi.org/10.5281/zenodo.3862368`

S1 Table. Description of agents, events, and operations that BioDynaMo currently provides.

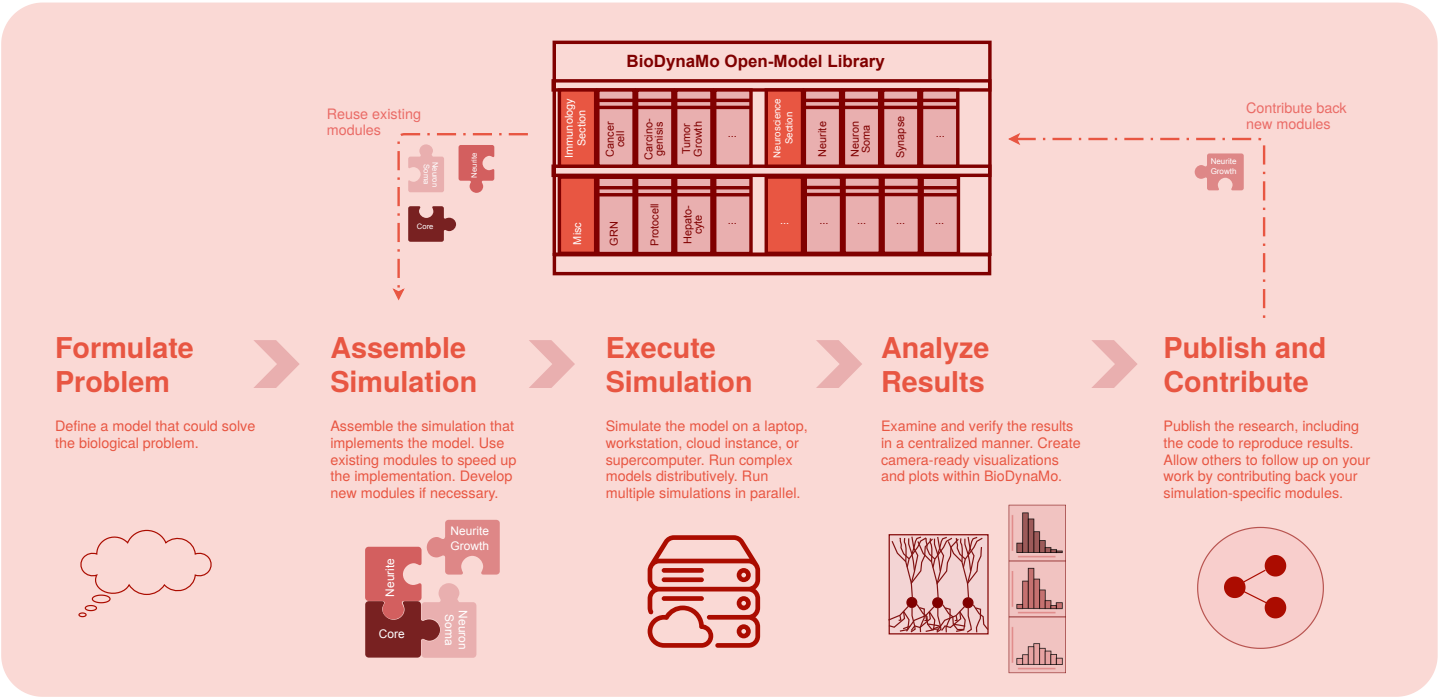

Fig 10. BioDynaMo platform. Vision of BioDynaMo, a platform to accelerate in silico experiments.

S1 File. Model parameters for the pyramidal cell and tumor spheroid growth simulation.

S2 File. 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. The README.md file provides instructions on how to execute the whole pipeline. Furthermore, it contains more details about the hardware and software configuration of the different systems described in Table 1.

S1 Video. Soma clustering simulation, as shown in Fig 5.
`https://www.youtube.com/watch?v=jlOk_Y3SUHo`

S2 Video. Single pyramidal cell growth simulation, as shown in Fig 6.
`https://www.youtube.com/watch?v=taWMFs5D5Pg`

S3 Video. Large-scale pyramidal cell growth simulation, as shown in Fig 7.
`https://www.youtube.com/watch?v=MA74wZbhO7w`

S4 Video. Tumor spheroid growth simulation, as shown in Fig 8.
`https://www.youtube.com/watch?v=Q9UkpLuLnkU`

Acknowledgments

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

References

1. Yankeelov TE, An G, Saut O, Luebeck EG, Popel AS, Ribba B, et al. Multi-scale Modeling in Clinical Oncology: Opportunities and Barriers to Success. Annals of Biomedical Engineering. 2016;44(9):2626–2641. doi:10.1007/s10439-016-1691-6.

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

3. Moore, Gordon E. Cramming More Components Onto Integrated Circuits. Electronics. 1965;38.

4. Dennard RH, Gaensslen FH, Rideout VL, Bassous E, LeBlanc AR. Design of ion-implanted MOSFET's with very small physical dimensions. IEEE Journal of Solid-State Circuits. 1974;9(5):256–268. doi:10.1109/JSSC.1974.1050511.

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

6. Railsback SF, Grimm V. Agent-based and individual-based modeling: a practical introduction. Princeton university press; 2019.

7. Azevedo FAC, Carvalho LRB, Grinberg LT, Farfel JM, Ferretti REL, Leite REP, et al. Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain. The Journal of Comparative Neurology. 2009;513(5):532–541. doi:10.1002/cne.21974.

8. Hucka M, Finney A, Sauro HM, Bolouri H, Doyle JC, Kitano H, et al. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. Bioinformatics. 2003;19(4):524–531. doi:10.1093/bioinformatics/btg015.

9. SBML Software Matrix; 2020. Available from: `http://sbml.org/SBML_Software_Guide/SBML_Software_Matrix`.

10. Emonet T, Macal CM, North MJ, Wickersham CE, Cluzel P. AgentCell: a digital single-cell assay for bacterial chemotaxis. Bioinformatics. 2005;21(11):2714–2721. doi:10.1093/bioinformatics/bti391.

11. Matyjaszkiewicz A, Fiore G, Annunziata F, Grierson CS, Savery NJ, Marucci L, et al. BSim 2.0: An Advanced Agent-Based Cell Simulator. ACS Synthetic Biology. 2017;doi:10.1021/acssynbio.7b00121.

12. Kang S, Kahan S, McDermott J, Flann N, Shmulevich I. Biocellion: accelerating computer simulation of multicellular biological system models. Bioinformatics. 2014;30(21):3101–3108. doi:10.1093/bioinformatics/btu498.

13. Koene RA, Tijms B, van Hees P, Postma F, de Ridder A, Ramakers GJA, et al. NETMORPH: A Framework for the Stochastic Generation of Large Scale Neuronal Networks With Realistic Neuron Morphologies. Neuroinformatics. 2009;7(3):195–210. doi:10.1007/s12021-009-9052-3.

14. Rudge TJ, Steiner PJ, Phillips A, Haseloff J. Computational Modeling of Synthetic Microbial Biofilms. ACS Synthetic Biology. 2012;1(8):345–352. doi:10.1021/sb300031n.

15. Torben-Nielsen B, De Schutter E. Context-aware modeling of neuronal morphologies. Frontiers in Neuroanatomy. 2014;8. doi:10.3389/fnana.2014.00092.

16. Ghaffarizadeh A, Heiland R, Friedman SH, Mumenthaler SM, Macklin P. PhysiCell: An open source physics-based cell simulator for 3-D multicellular systems. PLOS Computational Biology. 2018;14(2):e1005991. doi:10.1371/journal.pcbi.1005991.

17. Cytowski M, Szymanska Z. Large-Scale Parallel Simulations of 3D Cell Colony Dynamics. Computing in Science Engineering. 2014;16(5):86–95. doi:10.1109/MCSE.2014.2.

18. Lardon LA, Merkey BV, Martins S, Dötsch A, Picioreanu C, Kreft JU, et al. iDynoMiCS: next-generation individual-based modelling of biofilms. Environmental Microbiology. 2011;13(9):2416–2434. doi:10.1111/j.1462-2920.2011.02414.x.

19. Mirams GR, Arthurs CJ, Bernabeu MO, Bordas R, Cooper J, Corrias A, et al. Chaste: An Open Source C++ Library for Computational Physiology and Biology. PLOS Computational Biology. 2013;9(3). doi:10.1371/journal.pcbi.1002970.

20. Richmond P, Walker D, Coakley S, Romano D. High performance cellular level agent-based simulation with FLAME for the GPU. Briefings in Bioinformatics. 2010;11(3):334–347. doi:10.1093/bib/bbp073.

21. Pattana S. 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; 2006.

22. Sten-Knudsen O. In: Tosteson DC, editor. Passive Transport Processes. Berlin, Heidelberg: Springer Berlin Heidelberg; 1978. p. 5–113. Available from: `https://doi.org/10.1007/978-3-642-46370-9_2`.

23. Gurdon JB, Bourillot PY. Morphogen gradient interpretation. Nature. 2001;413(6858):797–803. doi:10.1038/35101500.

24. Bosch PS, Ziukaite R, Alexandre C, Basler K, Vincent JP. Dpp controls growth and patterning in Drosophila wing precursors through distinct modes of action. eLife. 2017;6:e22546. doi:10.7554/eLife.22546.

25. Smith GD, Smith GD, Smith GDS, Smither M, Press OU. Numerical Solution of Partial Differential Equations: Finite Difference Methods. Oxford applied mathematics and computing science series. Clarendon Press; 1985.

26. Brun R, Rademakers F. ROOT — An object oriented data analysis framework. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment. 1997;389(1):81–86. doi:10.1016/S0168-9002(97)00048-X.

27. de Montigny J, Iosif A, Breitwieser L, Manca M, Bauer R, Vavourakis V. An in silico hybrid continuum-/agent-based procedure to modelling cancer development: interrogating the interplay amongst glioma invasion, vascularity and necrosis. Methods. 2020;doi:https://doi.org/10.1016/j.ymeth.2020.01.006.

28. OpenMP Architecture Review Board. OpenMP Application Program Interface Version 4.5; 2015. Available from: `https://www.openmp.org/wp-content/uploads/openmp-4.5.pdf`.

29. Nvidia CUDA; 2020. Available from: `https://developer.nvidia.com/cuda-zone`.

30. OpenCL; 2020. Available from: https://www.khronos.org/opencl/.

31. Amdahl GM. 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). New York, NY, USA: ACM; 1967. p. 483–485. Available from: http://doi.acm.org/10.1145/1465482.1465560.

32. Ahrens J, Geveci B, Law C. ParaView: An End-User Tool for Large-Data Visualization. In: Visualization Handbook. Elsevier; 2005. p. 717–731. Available from: https://linkinghub.elsevier.com/retrieve/pii/B9780123875822500381.

33. Le Novère N, Bornstein B, Broicher A, Courtot M, Donizelli M, Dharuri H, et al. BioModels Database: a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems. Nucleic Acids Research. 2006;34(suppl_1):D689–D691. doi:10.1093/nar/gkj092.

34. Miller G. A Scientist's Nightmare: Software Problem Leads to Five Retractions. Science. 2006;314(5807):1856–1857. doi:10.1126/science.314.5807.1856.

35. Beck K, Gamma E. Extreme programming explained: embrace change. addison-wesley professional; 2000.

36. BioDynaMo Github repository; 2020. Available from: https://github.com/BioDynaMo/biodynamo.

37. Travis CI - Test and Deploy Your Code with Confidence; 2020. Available from: https://travis-ci.org/.

38. BioDynaMo Website; 2020. Available from: https://biodynamo.org/.

39. Mellström B, Kastanauskaite A, Knafo S, Gonzalez P, Dopazo XM, Ruiz-Nuño A, et al. Specific cytoarchitectureal changes in hippocampal subareas in daDREAM mice. Molecular Brain. 2016;9(1). doi:10.1186/s13041-016-0204-8.

40. Gong X, Lin C, Cheng J, Su J, Zhao H, Liu T, et al. Generation of Multicellular Tumor Spheroids with Microwell-Based Agarose Scaffolds for Drug Testing. PLOS ONE. 2015;10(6):1–18. doi:10.1371/journal.pone.0130348.

41. Nunes AS, Barros AS, Costa EC, Moreira AF, Correia IJ. 3D tumor spheroids as in vitro models to mimic in vivo human solid tumors resistance to therapeutic drugs. Biotechnology and Bioengineering. 2019;116(1):206–226. doi:10.1002/bit.26845.

42. Sutherland RL, Hall RE, Taylor IW. 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. 1983;43(9):3998–4006.

43. Macklin P, Edgerton ME, Thompson AM, Cristini V. Patient-calibrated agent-based modelling of ductal carcinoma in situ (DCIS): from microscopic measurements to macroscopic predictions of clinical progression. Journal of theoretical biology. 2012;301:122–140.

44. Altrock PM, Liu LL, Michor F. The mathematics of cancer: integrating quantitative models. Nature Reviews Cancer. 2015;15(12):730–745.