BioDynaMo: a general platform for scalable agent-based simulation

Lukas Breitwieser¹,²,*, Ahmad Hesam¹,³,*, Jean de Montigny¹, Vasileios Vavourakis⁴,⁵, Alexandros Iosif⁴, Jack Jennings⁶, Marcus Kaiser⁶,⁷,⁸, Marco Manca⁹, Alberto Di Meglio¹, Zaid Al-Ars³, Fons Rademakers¹, Onur Mutlu²,*, Roman Bauer¹⁰,*

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, o.mutlu@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
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 Wiletsky, 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; Cytwoski and Szymanska, 2014; Matyasjakiewicz et al., 2017; Ghaffariزاده 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 (Ghaffariزاده et al., 2018), Timothy (Cytwoski 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; Matyasjakiewicz et al., 2017; Rudge et al., 2012; Lardon et al., 2011), cell colonies (Kang et al., 2014; Mirams et al., 2013; Cytwoski 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; Cytwoski 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 opposite, 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.

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. Functionalties that are field-specific are separated from the core and are bundled as a separate module.

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
Dendritic development was dictated by a behavior defining growth parameters, a table with model parameters, and more detailed performance results in Supplementary File S1 Section 2.

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 (Züblin and Douglas, 2009), analyze its scalability, and quantify the impact of GPU acceleration for each use case.

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 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 dendritic’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
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 6000 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 mm³ 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 vivo study considered the same agarose gel matrix composition among the experiments, the BioDynaMo model assumes 100% transmission rate, and 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 some rate. The number of susceptible, infected, and recovered individuals, respectively, is the mean transmission rate, and the recovery rate. The example has 424 lines of C++ code, including the generation of 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 model the disease dynamics:

\[
\begin{align*}
\frac{dS}{dt} &= -\beta SI/N, \\
\frac{dI}{dt} &= \beta SI/N - \gamma I, \\
\frac{dR}{dt} &= \gamma I 
\end{align*}
\]

where \( S \) is the number of susceptible individuals, \( I \) is the number of infected individuals, \( R \) is the number of recovered individuals, \( N \) is the total number of individuals, \( \beta \) is the transmission rate, and \( \gamma \) is 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 some rate.

In this agent-based model, the speed at which an infectious disease spreads depends 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 overall 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 \( \beta \) and \( \gamma \) 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 \( \gamma \), 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).
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 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 Guangci Professorship Program of Ruijin Hospital (Shanghai University’s School of Computing [to J.J.]; the Wellcome Trust [102037 to R.B., MR/T004347/1 to M.K.]; the Engineering and Physical Sciences Research Council [to J.J.]; the ERC Starting Grants ‘Neurodis’ [617769 to R.B.]; the Israeli Government [to L.B. and A.H.]; the Israeli Innovation Authority [to A.H.]; the Research Council of Norway [to V.V.]; and the European Commission [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**


Mirams, G. R., Arthurs, C. J., Bernabe, M. O., Bordes, 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,
Breitwieser et al.


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:

\[ F_{ij} = k \delta - \gamma \sqrt{\delta} \]

where \( \delta \) indicates the spatial overlap between the two elements, and \( r \) denotes a combined measure of the two radii:

\[ r = \frac{r_1 r_2}{r_1 + r_2} \]

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 \( \gamma \). 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

\[
\begin{align*}
\mathcal{u}_{i,j,k}^{\nu+1} &= \mathcal{u}_{i,j,k}^{\nu} + \frac{\nu \Delta t}{\Delta x^2} \left( \mathcal{u}_{i+1,j,k}^{\nu} - 2 \mathcal{u}_{i,j,k}^{\nu} + \mathcal{u}_{i-1,j,k}^{\nu} \right) \\
&+ \frac{\nu \Delta t}{\Delta y^2} \left( \mathcal{u}_{i,j+1,k}^{\nu} - 2 \mathcal{u}_{i,j,k}^{\nu} + \mathcal{u}_{i,j-1,k}^{\nu} \right) \\
&+ \frac{\nu \Delta t}{\Delta z^2} \left( \mathcal{u}_{i,j,k+1}^{\nu} - 2 \mathcal{u}_{i,j,k}^{\nu} + \mathcal{u}_{i,j,k-1}^{\nu} \right) \times (1 - \mu),
\end{align*}
\]

where \( \mathcal{u}_{i,j,k}^{\nu} \) is the concentration value on grid point \((i, j, k)\) at timestep \( \nu \), \( \nu \) is the diffusion coefficient, \( \mu \) is the decay constant, \( \Delta t \) is the duration of one timestep, and \( \Delta x, \Delta y, \) and \( \Delta z \) 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 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.

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.

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

```cpp
1 2 3 4 5 6 7 8 9 10 11 12 13 14
BDM_AGENT_HEADER(Person, Cell, 1);
BDM_PARAM_GROUP_HEADER(SimParam, 1);
SimParam : { iterations = 100; };
Person : { state_ = State::kSusceptible; };
Listing 2: New agent class used in the epidemiology use case.

```cpp
1 2 3 4 5 6 7 8 9 10 11 12 13 14
class Person : public Cell { Public State { kSusceptible, kInfected, kRecovered; };
```

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.

Fig. 1. BioDynaMo notebook. A convenient web interface to create and run simulations in a step-by-step manner. The knitting 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.
of C++ code. Table 5 shows the performance on different systems. There such as oxygen or growth factors. The simulation consisted of 68 lines a benchmark, this example demonstrates the applicability of BioDynaMo 

Clusters were associated with non-homogeneous extracellular substance type. The final simulation state after 6000 time steps is shown in Figure 2B.

0 secretion_quantity the parameter secretion_quantity (Algorithm 6) and chemotaxis (Algorithm 7). We set 

Eq 3. We modeled cell processes with two behaviors, ran in sequence: 

cells. Substances diffused through the extracellular matrix following 

secreted a specific extracellular substance which attracted homotypic 

interaction between spherical cells, biological behavior, and cell division. 

The starting condition of this 

for our benchmarks and Table 5 lists the measured runtime and memory 

evaluation. Table 4 details the experimental environment that we used 

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Apical dendrite</th>
<th>Basal dendrite</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diameter threshold</td>
<td>0.575</td>
<td>0.75</td>
</tr>
<tr>
<td>Diameter threshold two</td>
<td>0.55</td>
<td></td>
</tr>
<tr>
<td>Old direction weight</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>Gradient weight</td>
<td>0.06</td>
<td>0.03</td>
</tr>
<tr>
<td>Randomness weight</td>
<td>0.3</td>
<td>0.4</td>
</tr>
<tr>
<td>Growth speed</td>
<td>100</td>
<td>50</td>
</tr>
<tr>
<td>Shrinkage</td>
<td>0.00071</td>
<td>0.00885</td>
</tr>
<tr>
<td>Branching probability</td>
<td>0.038</td>
<td>0.006</td>
</tr>
</tbody>
</table>

Algorithm 1: Apical and basal dendrite growth.

```cpp
1 auto Recovery : public Behavior |
2 { |
3 auto recovery = public Behavior |
4 |
5 void operator() override |
6 { |
7 auto* person = bdm_static_cast<Person*>(a); |
8 |
9 if (person->state_ == kInfected) |
10 { |
11 person->state_ = State::kRecovered; |
12 |
13 |
14 |
15 |
16 |
17 |
```

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

```cpp
1 auto TumorDiameterOp : public StandaloneOperationImpl |
2 { |
3 |
4 |
5 void operator() override |
6 { |
7 auto* sim = Simulation::GetActive(); |
8 |
9 |
10 |
11 |
12 |
13 |
14 |
15 |
16 |
17 |
```

Listing 4: New operation to determine the tumor spheroid diameter over time (oncology use case).
Algorithm 2: Cancer cell behavior.
\begin{verbatim}
input: cell, minimum_cell_age, death_probability,
   displacement_rate, growth_speed, division_probability
1 random_vector ← RandomUniform3(-1, 1);
2 brownian ← random_vector + random_vector.L2Norm();
3 cell.UpdatePosition(brownian × displacement_rate);
4 if age >= minimum_cell_age and
   RandomUniform(0, 1) < division_probability then
   5 cell.RemoveFromSimulation();
   6 return;
7 end
8 age ← age + 1;
9 if cell.GetDiameter < max_diameter then
10    cell.IncreaseVolume(growth_speed);
11 else if RandomUniform(0, 1) < death_probability then
12    cell.Divide();
13 end
end
\end{verbatim}

Algorithm 3: Infection behavior.
\begin{verbatim}
input: person, environment, infection_probability,
   infection_radius
1 if person.GetState() == susceptible and
   RandomUniform(0, 1) < infection_probability then
2   neighbors ← environment.GetNeighbors(infection_radius);
3   for each neighbor in neighbors do
4     if neighbor.GetState() == infected then
5        person.SetState(infected);
6     end
7   end
8 end
\end{verbatim}

Algorithm 4: Recovery behavior.
\begin{verbatim}
input: person, recovery_probability
1 if person.GetState() == infected and
   RandomUniform(0, 1) < recovery_probability then
2   person.SetState(recovered);
3 end
\end{verbatim}

Algorithm 5: Random movement behavior.
\begin{verbatim}
input: person, speed, max_bound
1 position ← person.GetPosition();
2 movement ← RandomUniform3(-1, 1).L2Norm();
3 new_position ← position + movement × speed;
4 for each el in new_position do
5    el ← fMod(el, max_bound);
6 if el < 0 then
7    el ← max_bound + el;
8 end
9 end
10 person.SetPosition(new_position);
\end{verbatim}

Algorithm 6: Soma clustering substance secretion.
\begin{verbatim}
input: cell, diffusion_grid, secretion_quantity
1 pos ← cell.GetPosition();
2 diffusion_grid.IncreaseConcentrationBy(pos, secretion_quantity);
\end{verbatim}

Algorithm 7: Soma clustering chemotaxis.
\begin{verbatim}
input: cell, diffusion_grid, gradient_weight
1 pos ← cell.GetPosition();
2 grad ← diffusion_grid.GetNormalizedGradient(pos);
3 cell.UpdatePosition(grad × gradient_weight);
\end{verbatim}

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
De
Problem
Formulate the biological problem.
fi
fine a model that could solve
for their simulation. Currently, we collect these contributions in our Github
providing the required building blocks (agents, biological behavior, etc.)
in Figure 4. This library will help scientists accelerate their research by
contributions will form the BioDynaMo open-model library, as shown
Fig. 4. BioDynaMo platform. Vision of BioDynaMo, a platform to accelerate in silico experiments.

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

<table>
<thead>
<tr>
<th>Parameter [dimensions]</th>
<th>2000 cells/well</th>
<th>4000 cells/well</th>
<th>8000 cells/well</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell growth rate [µm²/h]</td>
<td>42.0</td>
<td>35.0</td>
<td>29.9</td>
</tr>
<tr>
<td>Minimum cell age to apoptosis [h]</td>
<td>87</td>
<td>87</td>
<td>87</td>
</tr>
<tr>
<td>Division probability</td>
<td>0.0215</td>
<td>0.0215</td>
<td>0.0215</td>
</tr>
<tr>
<td>Cell death probability</td>
<td>0.033</td>
<td>0.033</td>
<td>0.033</td>
</tr>
<tr>
<td>Maximum cell speed [µm/h]</td>
<td>1.0</td>
<td>0.9</td>
<td>0.2</td>
</tr>
<tr>
<td>Cell-ECM adherence coefficient [dimensionless]</td>
<td>1.8</td>
<td>1.8</td>
<td>1.8</td>
</tr>
<tr>
<td>Random cell movement - displacement rate [µm/h]</td>
<td>0.005</td>
<td>0.005</td>
<td>0.0005</td>
</tr>
</tbody>
</table>

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.

SF3-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.

Table 3. Model parameters for the epidemiological use case.

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

collections 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).
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


### 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.

<table>
<thead>
<tr>
<th>System</th>
<th>Main memory</th>
<th>CPU / GPU</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>504 GB</td>
<td>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.</td>
<td>CentOS 7.9.2009</td>
</tr>
<tr>
<td>B</td>
<td>1008 GB</td>
<td>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.</td>
<td>CentOS 7.7.1908</td>
</tr>
<tr>
<td>C</td>
<td>191 GB</td>
<td>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.</td>
<td>Ubuntu 20.04.1 LTS</td>
</tr>
<tr>
<td>D</td>
<td>16 GB</td>
<td>One Intel(R) Core(TM) i7-7600U CPU @ 2.80GHz with two physical cores.</td>
<td></td>
</tr>
</tbody>
</table>

### 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.

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