Journal of Molecular Graphics and Modelling 68 (2016) 68–77

Contents lists available at ScienceDirect

Journal of Molecular Graphics and Modelling

journal homepage: www.elsevier.com/locate/JMGM

High performance computing for three-dimensional agent-based molecular models

G. Pérez-Rodríguez[a], M. Pérez-Pérez[a], F. Fdez-Riverola[a], A. Lourenço[a,b,*]

[a] ESEI – Escuela Superior de Ingeniería Informática, Edificio Politécnico, Campus Universitario As Lagoas s/n, Universidad de Vigo, 32004 Ourense, Spain
[b] CEB – Centre of Biological Engineering, University of Minho, Campus de Gualtar, 4710-0668 Braga, Portugal

ARTICLE INFO

Article history:
Received 23 November 2015
Received in revised form 26 May 2016
Accepted 7 June 2016
Available online 8 June 2016

Keywords:
Cellular systems
Agent-based modelling
Three-dimensional model representation
Individual molecule simulation
Distributed computing

ABSTRACT

Agent-based simulations are increasingly popular in exploring and understanding cellular systems, but the natural complexity of these systems and the desire to grasp different modelling levels demand cost-effective simulation strategies and tools.

In this context, the present paper introduces novel sequential and distributed approaches for the three-dimensional agent-based simulation of individual molecules in cellular events. These approaches are able to describe the dimensions and position of the molecules with high accuracy and thus, study the critical effect of spatial distribution on cellular events. Moreover, two of the approaches allow multi-thread high performance simulations, distributing the three-dimensional model in a platform independent and computationally efficient way.

Evaluation addressed the reproduction of molecular scenarios and different scalability aspects of agent creation and agent interaction. The three approaches simulate common biophysical and biochemical laws faithfully. The distributed approaches show improved performance when dealing with large agent populations while the sequential approach is better suited for small to medium size agent populations.

Overall, the main new contribution of the approaches is the ability to simulate three-dimensional agent-based models at the molecular level with reduced implementation effort and moderate-level computational capacity. Since these approaches have a generic design, they have the major potential of being used in any event-driven agent-based tool.

1. Introduction and motivation

New advances in super-resolution and super-localisation techniques have allowed experimental molecular biophysics and biochemistry to go beyond ensemble measurements and obtain data at the single molecule level [4,15]. Such experiments are able to track down key motions, reactions, and interactions of individual molecules with high temporal and spatial resolution. However, the acquisition of such data is very time-consuming, partially because the techniques are not yet advanced enough to allow the simultaneous observation of a wide range of molecule types [26,30].

An alternative to the use of experimental techniques is to assemble molecular models *in silico* and to use simulation techniques to explore their behaviour. Such computational models have the potential to elucidate structure and auto-organisation between molecules as well as complex molecular interplay that are difficult to observe *in vivo* or *in vitro*. In this context, the challenge presented to computational methodologies is to embrace the natural complexity of cellular systems as faithfully as possible [51]. A realistic model, depending on the cellular system at hand and the questions to be asked, should cope with spatial and temporal scales of various orders of magnitude and different levels of modelling detail. Biologically relevant time scales range from nanoseconds to microseconds for the internal dynamics of individual molecules; cellular dimensions are between 300 nm for the smallest bacterial cells and 100 μm for large eukaryotic cells; and, the atomistic structural description of molecules requires spatial resolution in the nanometre range [14]. Moreover, it is important to model the molecules and the environment as volumes in order to have full awareness of the spatial location of the molecules, and the implications over biophysics (e.g. collisions) and biochemical rules (e.g. reaction radius) [33].

So, although it is conceivable to model cellular processes at single molecule level, such fine grain simulation demands

* Corresponding author at: ESEI – Department of Computer Science, Campus Universitario As Lagoas s/n, 32004 Ourense, Spain.
E-mail addresses: gprodriguez2@esei.uvigo.es (G. Pérez-Rodríguez), mpperez3@esei.uvigo.es (M. Pérez-Pérez), riverola@uvigo.es (F. Fdez-Riverola), analia@uvigo.es (A. Lourenço).

http://dx.doi.org/10.1016/j.jmgm.2016.06.001
1093-3263/© 2016 Elsevier Inc. All rights reserved.

considerable computing power. In this context, general purpose Graphical Processing Unit (GPU) technology and multi-core CPU processors are being used to parallelise biomolecular simulations [20,21,23,40]. However, such frameworks do not yet support three-dimensional modelling and are not meant to be deployed in general biological research settings, specifically in research centres or labs that do not have access to high performance computing clusters, parallel architectures and GPU hardware, nor have the technical expertise to write efficient software for these environments. So, it is often the case that computational biologists resort to coarser resolution approaches to simulate large biological systems [41,43]. Coarser models keep a reduced and essential number of degrees of freedom and interactions, which decreases the computational requirements of the simulation and allows the execution of simulations in computers with moderate computational capacity. Still, this reduction can go so far and, in practice, single cell coarse models also demand the implementation of simulation optimisation algorithms.

In this context, this work aims to contribute to the simulation of individual molecules in cellular events in two ways: to enable the realistic simulation of the spatial location, diffusion and interaction of molecules in three-dimensional, continuous environments; and, to improve the generic template of agent-based approaches so that the computational costs of such realistic simulations are affordable. Most notably, the aim is to enable realistic simulation in computers with a moderate-level computational capacity, i.e. computers broadly available in biological labs and that do not required advanced programming skills. Hence, our work proposes solutions that may be applied to virtually any model at the molecular level and may be used in any event-driven agent-based tool.

The next sections describe the capabilities of existing ABM software for biomolecular simulation, the solutions devised for scalable three-dimensional single molecule simulation, and the analysis of performance results for the proposed approaches.

2. Related work: existing approaches for biomolecular modelling and simulation

Agent-based models (ABM) are a well-known and favoured modelling strategy for biomolecular systems [7]. Generically, these models are composed by a population of heterogeneous agents, which represent the molecules under study, including their shape, size and interaction logic. Biomolecular events unfold on an explicit and specific environment (e.g. representing the cytoplasmic environment) where agents act autonomously, executing some sort of itinerary (e.g. molecular diffusion). Each agent class is implemented to represent the features and behavioural responses of a specific type of molecules (e.g. enzymes and metabolites). Agents interact with one another following common biochemical and biophysical assumptions (e.g. enzymatic kinetics), and their behaviour may adapt depending on the perceived situation, and most notably the influence of the immediate surroundings (e.g. abundance of substrate).

2.1. ABM for biomolecular modelling

Individual particle or agent-based modelling is one of the current favoured alternatives for biomolecular representation, as it is naturally suited to describe single molecule behaviour and help understanding phenotypic heterogeneity and cell-to-cell variability.

Among the earliest biomolecular models one may find Cellulat, an agent-based intracellular signalling model [18], and Epitheliome, which represents the growth and repair characteristics of epithelial cell populations [49]. Today, the range of agent-based model applications is quite broad. For example, these models have been used to represent the Ras–MAPK [17] and NF-kB [38] intracellular signalling pathways, *Escherichia coli* cytoplasm dynamics [30], bacterial phenotypic switching [46], epithelial host-pathogen interactions [45], cancer systems biology [29], development of restenosis in blood vessels [12], autophagy dynamics and sub-mitochondrial heterogeneity [8], intracellular phosphorus heterogeneity in cultured phytoplankton [16], oxygen metabolism in aerobic-anaerobic respiration [5], and the design of cellulase systems [3].

A deep description of general agent-based modelling approaches and common "recipes" used in biomolecular modelling are out of the scope of this work. We recommend reading the following state-of-the-art reviews for gaining further understanding about the general aspects of ABM and computational social science [7,10], the use of ABM to model biological complexity across biological scales [1,22], the construction of biological ABM under the Systems Biology perspective [31,34] and existing models of biomolecules in cellular environments [14].

2.2. Software platforms for ABM simulation

Commonly, ABM software follows the framework and library paradigm, i.e. a framework that provides a set of standard concepts for designing and describing ABMs, and a software library implementing the framework and providing simulation tools. Swarm was one of the first ABM software [19]. It was written in Objective-C and served as inspiration to most of the more recent software. The well-known Recursive Porous Agent Simulation Toolkit (Repast) started as a Java implementation of Swarm, but evolved on its own [35]. The project has released several software toolkits and development environments (in Java, Python and NET) and recently Repast has been superseded by a significant development named Repast Simphony, or Repast-S [36]. The Multi-Agent Simulation of Neighbourhoods (MASON) is another prominent toolkit, which was designed as a smaller and faster alternative to Repast, focused on computationally demanding models with many agents executed over many iterations [27].

Alternative development exists and NetLogo, from the Logo family of platforms, is one of the best representatives [44]. Here, the purpose is mainly educational, more specifically to provide a high-level platform that allows non-skilled users to build and learn from simple ABMs. Nevertheless, the platform now contains many sophisticated capabilities (behaviours, agent lists, graphical interfaces, etc.) and it has been used in biomedical applications.

Throughout the years, several studies have analysed and compared the evolution of these platforms in terms of conceptual basis, programming experience, execution speed, development priorities, and ease of use, i.e. the intricacy of implementing ABMs with them [25,28,39,47]. NetLogo stands out for models that are compatible with its paradigm of short-term, local interaction of agents and a grid environment and are not extremely complex. Java Swarm is outperformed by more recent, alternative Java platforms. Overall, although current multi-agent platforms demonstrate ability for modelling and understanding phenomena of increasing complexity, new releases and new platforms keep emerging. So, it is not straightforward to choose a platform, in particular to those looking to develop domain-specific software. This decision is usually left to survey articles, past domain application experiences and platform publicity.

2.3. Biomolecular-specific agent-based simulation

In the case of biomolecular simulation, one may observe that usually development does not rely on general ABM platforms. Well-established biomolecular simulators such as ReaDDy [42], Smoldyn

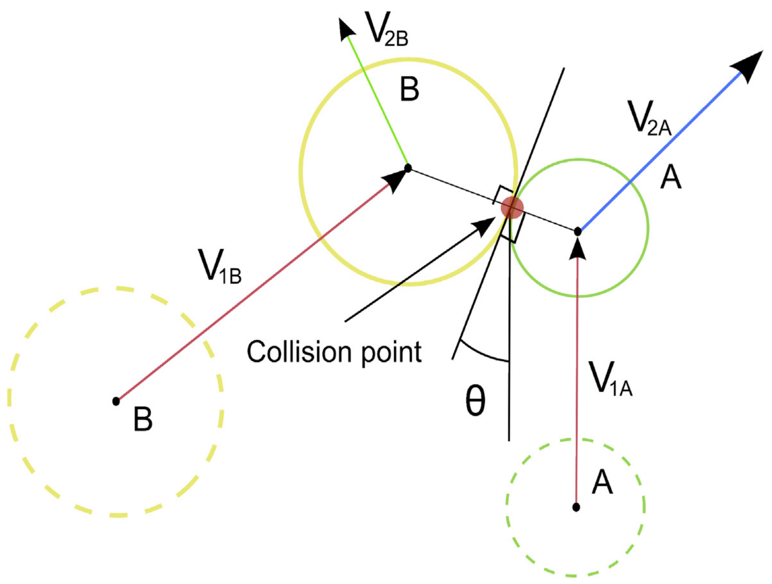

Fig. 1. Example of collision between two different molecules: enzyme (A) and substrate (B). V_{1A} and V_{1B} stand for the initial velocity vectors of each molecule whilst V_{2A} and V_{2B} represent the final corresponding counterparts. θ is the angle between the normal and the actual velocity vector of each agent.

[2], Klann's simulator [24], and the Cellular Dynamic Simulator [6] present specialised particle-based designs for three-dimensional continuous simulation.

Arguably, the greatest impediment to a broader use of ABM frameworks in biomolecular simulation is the complexity of the scenarios to be simulated, which involve a large number of heterogeneous agents interacting on the basis of numerous biophysics and biochemical assumptions, and demand skilled and cost-effective programming. Recently, different reviews have discussed and compared available biomolecular simulators [13,22,43]. The computational costs associated to the simulation of increasingly complex models are critical to current development. Researchers aim to simulate more realistic scenarios, more specifically three-dimensional and continuous cellular environments that include a large number of molecules of different form and size.

Many existing biological specific simulators still operate on two-dimensional grid-based environments and with point-like particles, and the ones that already support more realistic features present a proprietary architecture, which requires specialised (i.e. highly architecture dependent) computational optimisation or extension efforts. In turn, most of the current ABM frameworks (such as FLAME, REPAST or MASON) can operate in three dimensional environments and have the flexibility to implement high performance computing strategies. Some of these frameworks offer distributed solutions over a cluster of computers (such as REPAST HPC [9] and D-MASON [11]) or GPU-based (such as FLAME GPU [40]) that allow for more computational power to run complex models.

In the present work the goal was to propose computing approaches that could support realistic biomolecular simulations in general-purpose computers, i.e. computers with a moderate-level computational capacity. Our programming efforts were focused on implementing three-dimensional continuous environments and enhancing performance to enable realistic simulation in a non-specialised way. Therefore, we did not consider any frameworks that took advantage of clusters or GPU technology. Likewise, general purpose frameworks were preferred over specific simulators as means to ensure that the approaches could be used by a larger scope of users and applications. Finally, we decided to use MASON over other frameworks based on the number of users that cite the use of this framework in their scientific publications (according to Google Scholar). However, our approaches are not MASON-dependent. MASON is used here with the intent of showing that our approaches are able to improve simulation performance without modifying the data structures and algorithms existing in general purpose frameworks. Therefore, our approaches hold the potential of being used in eventually any event-driven simulator.

3. High performance approaches for three-dimensional agent-based biomolecular simulation

Generally, a three-dimensional model simulated at the molecular level should describe: the volume, shape, localisation, direction vector and speed of each molecule; the rules of interaction between molecules; and, the volume of the environment (e.g. the cytoplasm, a growth volume, or the cell).

Therefore, the implementation of our biomolecular model in MASON entailed the definition of common biophysics and biochemical laws and assumptions. More specifically, our model accounts for molecular diffusion [48] and collision resolution (illustrated in Fig. 1) as basics means to guarantee that agent movement complies with the Brownian motion of molecules. Also, the behaviour logic of our model is consistent with well-known molecular interactions [32,37].

For example, the behavioural rules describing an enzymatic reaction establish that the agent representing the enzyme E should bind to the agent representing the substrate S, whenever S is in the vicinity of E (i.e. within the enzyme radius). As represented in Expression (1), this binding leads to a catalytic event and, as a result, the agent S will die and a new agent representing the product P will be created.

$$E + S \overset{Binding}{\leftrightarrow} ES \overset{Catalysis}{\rightarrow} E + P \tag{1}$$

The binding and catalysis events are modelled by two parameters: K_m, which defines the affinity of the enzyme towards the substrate and k_{cat}, which quantifies the probability of occurrence of a successful collision between the enzyme and substrate [50].

In this context, MASON offered an appropriate baseline, single-threaded approach for the simulation of our biomolecular model (introduced in Section 3.1). From this starting point, we then developed two high performance alternatives: a parallel approach

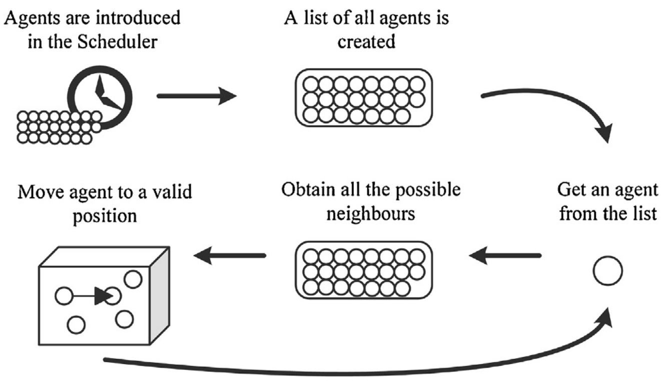
Agents are introduced in the Scheduler
A list of all agents is created
Get an agent from the list
Obtain all the possible neighbours
Move agent to a valid position

Fig. 2. Single-thread ABM simulation using MASON.

(introduced in Section 3.2) and a partitioned approach (introduced in Section 3.3).

3.1. *Single-thread MASON approach*

Typically, MASON operates in single-thread mode, i.e. its scheduler keeps track of all the agents in the model and ensures their sequential execution. Therefore, if the model requires the simulation of 10,000 molecules of substrate, the scheduler manages a list formed by 10,000 agents to be executed one by one (see Fig. 2).

Simulation starts by iteratively positioning all the agents. The specific location of each agent can be defined by the user or be randomly calculated by MASON, but in any case the position has to be validated (check if the position is vacant) and, if necessary, it has to be re-calculated until a valid position is obtained. This operation is repeated until all the agents are correctly positioned or the environment is crowded.

At each simulation time step, and in a sequential manner, every agent looks for its immediate neighbours in the list of total agents (in this case, those agents which are within twice the agent's radius range) and checks if any of its rules (encoding its behaviour) may be applied, triggering the execution of an event. Our model supports three different types of events: (*i*) *reactions*, which include bind, elimination and creation of new agents; (*ii*) *rebound*, which implies a change in the direction of those agents involved in a collision; and (*iii*) *movement*, which is the default action carried out by the agent if none of the previous events can be applied and the new estimated position is vacant.

This elementary operation mode suffers from an obvious performance problem on large agent populations. Performance will be affected by the need to evaluate the logic of each agent (i.e. associated behavioural rules and molecular diffusion) at each simulation step.

3.2. *Parallel approach*

An alternative to single-thread simulation is multi-thread execution based on the creation and management of a shared population of agents. That is, several threads iterate the list of agents and thus multiple agents may execute their associated logic simultaneously.

As illustrated in Fig. 3, and in contrast to MASON single-thread operation, our parallel approach introduces two important changes: (*i*) it introduces only one agent in the MASON scheduler, the controller, which, at simulation start, creates the different threads that will handle the population of agents; (*ii*) it uses an optimised data structure to look for neighbours during an agent step. This structure is a map that stores the agents by their position in one of the three axes (x, by default). Thanks to this, it is only necessary to iterate the agents that are far away from the centre ± radius instead of iterating the complete list of agents. The number of threads can be defined by the user, but a good initial approximation is using as least as many threads as cores are available in the host machine.

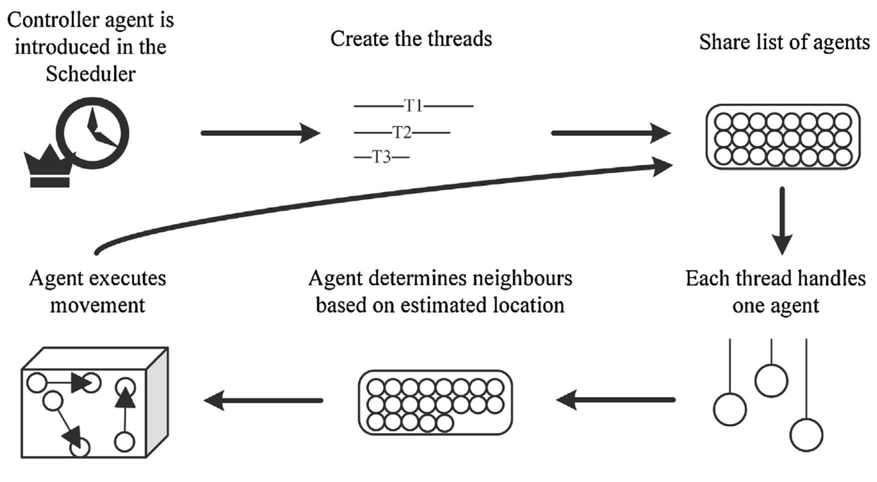
Controller agent is introduced in the Scheduler
Create the threads
T1
T2
T3
Share list of agents
Each thread handles one agent
Agent determines neighbours based on estimated location
Agent executes movement

Fig. 3. Parallel ABM alternative to carry out biomolecular simulations in MASON.

```
Iterate shared list of agents
    Pick up an agent
    Estimate its new position (movement)
    Evaluate its behavioural rules (event)

Wait in the barrier for the rest of agents (synchronization)

Iterate shared list of agents
    Pick up an agent
    Execute its pending actions (movement)
```

Fig. 4. Code snippet showing the list of actions that each thread should execute during a time step in the parallel approach.

At simulation time steps, the controller is responsible for managing the shared list of agents and ensuring a valid execution of the agents by the threads. Thread execution follows a two-step algorithm (see pseudocode in Fig. 4). In the first step, the thread picks one agent from the shared list to be executed (line 2). More specifically, the agent calculates its potential next movement based on its diffusion rate and direction of movement (line 3). Moreover, it checks if any of its behavioural rules may be applied and, if so, a new event is executed (line 4). As explained in the single-thread approach, one of three possible events can take place: (*i*) an enzymatic reaction, i.e. an enzyme agent and the corresponding substrate agent are within a reactive vicinity, so the two agents bind and, as a result, the substrate is consumed and a new product agent is created; (*ii*) a collision is detected and the agent bounces, i.e. recalculates its direction of movement; or, (*iii*) the previously estimated position is vacant and none of the previous events has occurred. If a reaction takes place, the new agents (i.e. product agent and free enzyme agent) will be created in the position of the biggest input agent. If a rebound occurs, the agent only changes direction, but it does not move. Finally, if the only option is to move and the estimated position is vacant, the next event of the agent will be to move. However, movements have to be synchronised, i.e. an agent cannot move while other agents are still determining their new position or evaluating their behavioural rules (all agents are entitled to only one event per time step).

In this regard, the function of the barrier is to synchronize all the agents in the simulation (line 6). Afterwards, in the second step of thread execution, agents will execute the pending movements (lines 8–10).

Thread synchronisation is critical in this approach because two threads cannot access the same agent simultaneously. To prevent this situation, our approach implements agent locks and a complementary maximum priority thread to aid in conflict resolution. Every time a thread picks an agent from the shared list of agents, the agent is locked. While executing the logic of the agent, more specifically looking into the agent's neighbourhood, the thread may try to access an agent locked by another thread (information retrieved by consulting the state of the agents). That is, the thread will dispute a resource that is busy. If this thread tie is not broken in a limited (preconfigured) number of attempts, the first thread reaching the threshold will unlock its current agent (re-inserted in the shared list of agents) and pick another agent from the shared list. The other thread may then lock the agent and complete the desired actions. In the situation that the two threads reach the threshold at the same time, both agents will be released without having performed any action. The maximum priority thread will pick these agents and guarantee that they will be executed.

Fig. 5 introduces an illustrative example of thread synchronisation. At instant *t* (left side panel), thread 1 locks agent 1 (in yellow) while thread 2 locks agent 2 (in orange). When agent 1 inspects its neighbourhood (looking into the shared list of agents) it detects that agent 2 is inside its area of influence (big yellow dotted circle), and same happens when agent 2 looks into its neighbourhood. So, the two threads will dispute both agents. In our example, the tie is resolved by thread 2 releasing agent 2 and picking up another agent from the shared list of agents (agent 3 in blue). A similar situation can be observed for agents 4 and 5 (in red and green, respectively).

At instant *t+1* (right side panel), we can observe how all molecules have executed their actions successfully: movement (agents 1, 2 and 3) and rebound (agents 4 and 5). In the next time step (not shown), agent 3 and its closest neighbour (in grey) are going to collide and the grey agents at the top right of the figure are going to bounce off the boundary of the simulation environment.

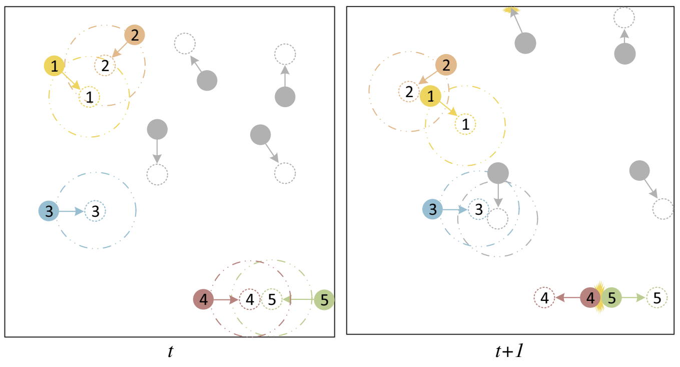

Fig. 5. Theoretical example of possible synchronization situations handled by the parallel approach. Scenario involves 5 different agents and two consecutive time steps. (For interpretation of the references to colour in the text, the reader is referred to the web version of this article.)

3.3. Spatial partitioning approach

With the particular goal of optimising, even more, the repetitive calculation of agent neighbours in simulations with large populations of agents, we have designed a spatial partitioning approach. This approach divides the three-dimensional continuous simulation environment into smaller, partially autonomous subspaces (partitions).

Similarly to the parallel approach, our partitioning approach introduces only one agent in the MASON scheduler: the controller (see Fig. 6). At simulation start, this agent creates the environment partitions, each to be executed by a different thread. In this approach we maintain the same optimised data structure to check neighbours. However, each partition manages its own map with the corresponding subset agents, which decreases the number of iterations. The number of partitions can be specified by the user, but a good initial approximation is to consider as many partitions as cores are available in the host machine. Noteworthy, the number of partitions is tightly bound to population distribution, i.e. a higher number of partitions will increase the performance when the number of agents per partition is balanced. However, situations where the distribution of agents is unbalanced can be further solved by implementing a load balance strategy for this approach (e.g., resizing the partitions dynamically depending on the number of agents in each partition).

As described by the pseudocode in Fig. 7, each partition has its own population of agents and can execute the logic associated to these agents in a semi-autonomous way. Exception being the agents located in fringe (boundary) regions (called transition agents), which need to check for neighbours in contiguous partitions and therefore, require synchronisation among partitions. So, each agent in the partition will estimate its new possible position, if it is in a fringe region the agent is inserted in the transition agent list, otherwise the agent may evaluate its behavioural rules (lines 3–8).

The barrier is used to ensure that all agents have executed an event or determined their next position before allowing further movement (lines 10 and 16). Similarly to the parallel approach, the synchronised execution of the agents in the transition list is ensured by a maximum priority thread (lines 12–14). As such, the algorithm prevents new conflicts when resolving the state of transitioning agents (e.g. when the maximum priority thread looks for neighbours across partitions), because all non-transitioning agents are idle (in the barrier). Afterwards, agents will execute the pending movements (lines 18–20).

Fig. 8 exemplifies different scenarios of agent transitions in four partitions, each one managed by a different thread (i.e. T1, T2, T3

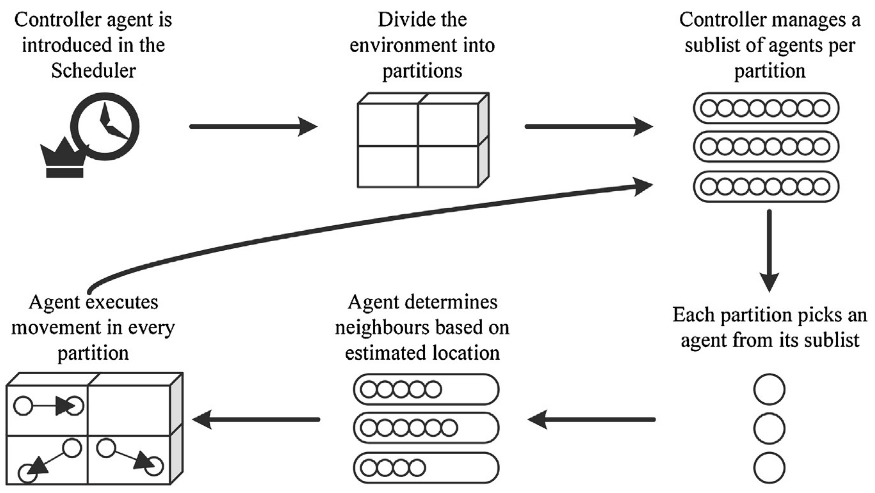
Controller agent is introduced in the Scheduler
Divide the environment into partitions
Controller manages a sublist of agents per partition
Each partition picks an agent from its sublist
Agent determines neighbours based on estimated location
Agent executes movement in every partition

Fig. 6. Spatial partitioning ABM approach to carry out biomolecular simulations in MASON.

```
Iterate sub list of agents
    Pick up an agent
    Estimate its new position (movement)
    IF the agent is transitioning THEN
        Put the agent in the transition list
    ELSE
        Evaluate its associated behavioural rules (event)
    ENDIF

Wait in the barrier for the rest of agents (synchronization)

Iterate transition list (by a maximum priority thread)
    Pick up an agent
    Evaluate its associated behavioural rules (event)

Wait in the barrier for the rest of agents (synchronization)

Iterate sub list of agents
    Pick up an agent
    Execute its pending actions (movement)
```

Fig. 7. Code snippet showing the list of actions that each thread should execute during a time step in the spatial partitioning approach.

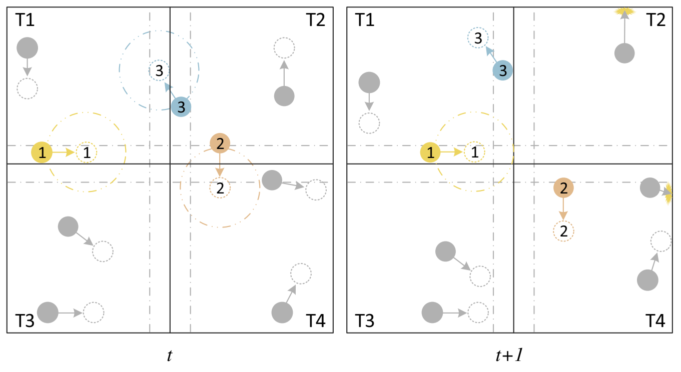

Fig. 8. Theoretical example of the functioning of four partitions, containing several agents, in two consecutive timesteps.

and T4). At instant *t* (left side panel), all partitions can calculate the events to be executed (i.e. reaction, rebound or movement) for the grey agents inside their subspaces, because they are not in the boundary region (dotter light grey line). However, in the subspaces of partition 1 and partition 2 there are agents in the boundary region (agent 1 in yellow for partition 1 and agents 2 and 3 in orange and blue, respectively, for partition 2). As explained before, all these transition agents will be handled by the maximum priority thread whereas the grey agents will be executed autonomously, within the scope of the corresponding partition.

At instant *t+1* (right side panel), one can observe that agents have executed their associated events (movement) and, most notably, some of the transition agents have crossed from one partition to another. Agent 1 (in yellow) remains in a boundary region and thus, needs to check again its neighbours in partition 1 and 3. In turn, two grey agents are going to bounce off the upper and right boundaries of their subspaces (in the second and fourth partitions, respectively).

Although this spatial partitioning approach introduces the need to synchronize adjacent partitions to account for events in the boundary regions, this synchronization is less expensive than

Table 1
Description of the biomolecular model used as case study.

Molecule	Weight (Da)	Radius (nm)	Diffusion (μm2/s)	Concentration (mM)	K_m (mM)	K_{cat} (s^{-1})
Enzyme	22500	2.62	125	1.22×10^{-2}	0.1449	1.39×10^{6}
Substract	158.11	0.93	353	6.09×10^{-2} to 2.44×10^{1}	–	–
Product	158.11	0.93	353	–	–	–

the synchronisation necessary in the parallel approach. There is no need to maintain a lock in every agent due to the partially autonomous nature of each partition. Every partition has its own list of agents managed by its own thread. The maximum priority synchronisation thread only deals with the transition agents, as explained before.

4. Results and discussion

The simulation of the enzymatic activity of 2-hydroxymuconate tautomerase (EC 5.3.2.6) was used to compare the proposed approaches. The three-dimensional continuous environment was dimensioned for a volume of 0.48 μm^3. Table 1 details molecule characteristics, namely dimensions (i.e. molecular weight and radius), rate of movement (molecular diffusion) and number (concentration). The enzyme is also characterised by the kinetic parameters K_m and k_{cat}. Further details on this enzyme and its activity can be found at BRENDA enzyme database Chang et al., 2014.

Considering different initial concentrations of substrate, it is possible to determine whether the rate of product generated in the simulations is consistent with the theoretical, to be expected values. Fig. 9 presents a snapshot of these simulations and two complementary plots comparing *in silico* and theoretical results. The snapshot in Fig. 9a illustrates the population of enzymes (yellow coloured spheres), substrates (green coloured spheres) and products (orange coloured spheres) at a given time step. The plot in Fig. 9b represents the Michaelis-Menten saturation curves for the theoretical results and the *in silico* results. For the parallel and partitioned approaches we computed 20 independent runs with the corresponding average and deviation represented by box-and-whisker plots. Complementarily, the plot in Fig. 9c shows the Lineweaver and Burk linear transformations of the saturation curves and the corresponding trend lines based on linear regression.

From a biological point of view, the evaluation procedure involved revising the model and the rules guiding the behaviour of the simulation till the predicted data were in the same order of magnitude of the values of Michaelis-Menten parameters (Fig. 9). A smaller deviation could still be achieved by fine tuning the reaction radius parameter, in an iterative fashion, but this is out of the scope of the present work. Details on model validation can be found in Supplementary material S1.

The next sections discuss the performance of the three proposed approaches considering the above described enzyme model. The performance of the approaches is compared in terms of the most critical stages in the execution of the biomolecular simulation: the creation of agents and their repositioning in simulation steps. Additionally, we analyse the influence of the number of available threads and the dimensions of the environment over the performance of the distributed approaches. Tests were performed on a computer with an Intel CPU I7 860 @ 2.80 GHz and 8 GB of RAM DDR3 @ 1333 MHz CL9 running Windows 7 Professional 64 bits.

4.1. Agent creation

At agent creation, computational costs are mainly associated with the calculation of a valid initial position for each agent. Therefore, our evaluation considered a growing population of agents (from 0 to 100,000 agents) and monitored the duration of the agent creation operation in the three simulation approaches.

As shown in Fig. 10, the sequential approach (dash dotted line) outperforms the multithread (dashed line) and partition (solid line) alternatives for small-medium populations of agents (in the current setup, less than 25,000 agents). In such a situation, the sequential

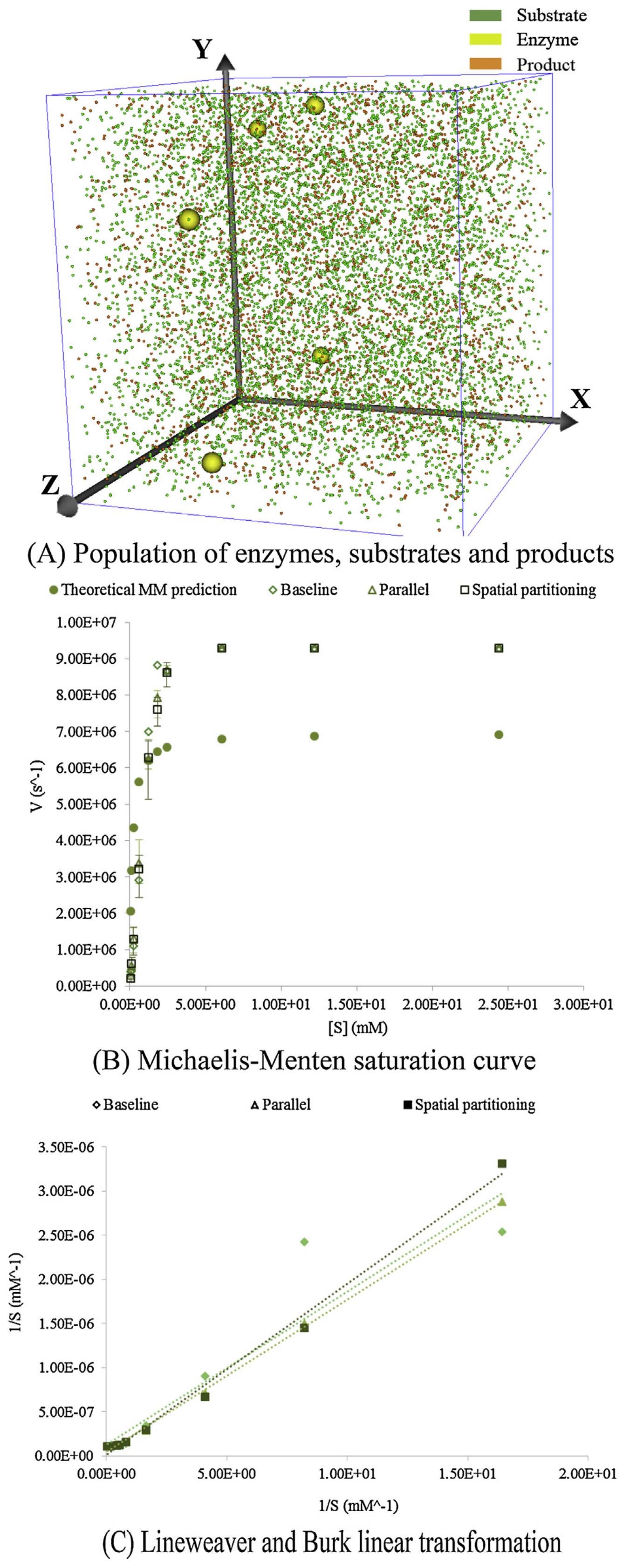

Fig. 9. Data obtained from the execution of the biomolecular model describing the enzymatic activity of 2-hydroxymuconate tautomerase. (A) Snapshot of the simulation environment during the execution of the simulation. (B) Michaelis-Menten saturation curves representing the average results of the runs of the three approaches and the corresponding theoretical values. (C) Lineweaver and Burk linear transformation of the Michaelis-Menten saturation curves.

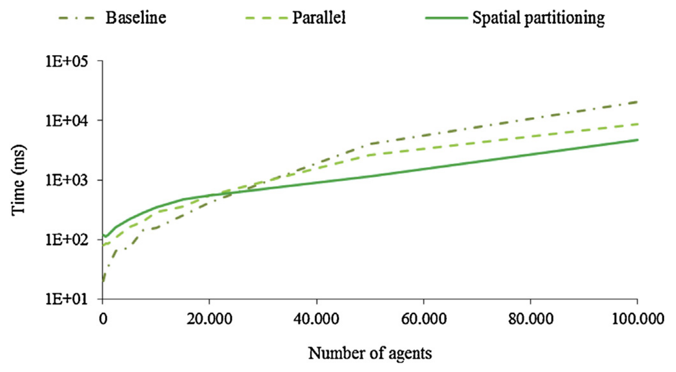

Fig. 10. Performance of the simulation approaches during agent creation.

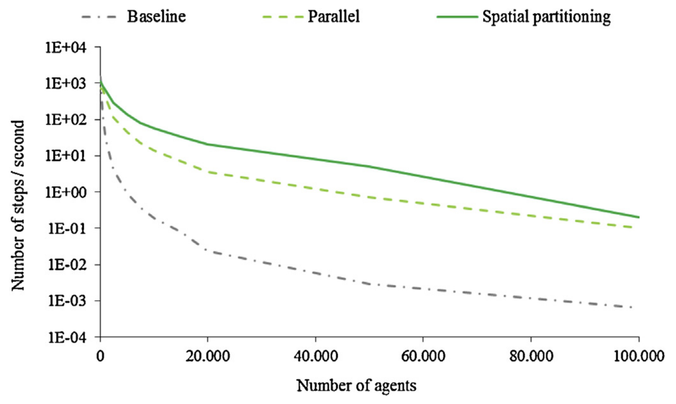

Fig. 11. Performance of the simulation approaches during agent execution. The synchronization time in distributed approaches is also included.

schema is faster than the distributed approaches because the latter need additional resources to handle agent synchronisation.

However, for larger populations of agents, the average performance achieved by the distributed approaches is better than the performance of the sequential approach. The reason is that in the sequential simulation, the list of agents has to be checked entry by entry every time a new position is calculated, i.e. the bigger the list of agents to be checked the more time the operation will take. Since the distributed approaches use an optimised data structure based on agent positions they are available to improve this calculation. In fact, the cost of synchronising the distributed operations is diluted by the ability to cope with position checking in a more efficient way (i.e. inserting in a distributed way and iterating only the most nearest neighbours using the data structure). Notably, the spatial partitioning approach presents the best average of computation time. Further details on these experiments can be found in Supplementary material S2.

4.2. Agent execution

The rationale of comparing the time taken by the different approaches to execute a simulation time step is to observe how the available alternatives deal with the logic of the agents and perform their repositioning (molecular diffusion). Therefore, our experiments considered a growing population of agents (from 0 to 100,000 agents) and monitored the number of simulation steps executed per second (see Fig. 11).

As illustrated in Fig. 11, the performance of the sequential approach (dash dotted line) quickly deteriorates when dealing with larger populations and both the parallel (dashed line) and spatial partitioning (solid line) approaches clearly outperform its performance. Compared to the baseline, the parallel and spatial approaches have an average performance increment of 121% and 189%, respectively. While in the sequential approach (baseline performance) the agents need to be positioned one by one, the distributed approaches rely on optimised position search structures and a parallel execution of multiple agents. More specifically, the parallel approach stores agents in a shared, sorted and position-based map and the spatial partitioning approach implements a similar structure per partition (whereas considering neighbours from other partitions for agents in border regions). Further details on these experiments can be found in Supplementary material S3.

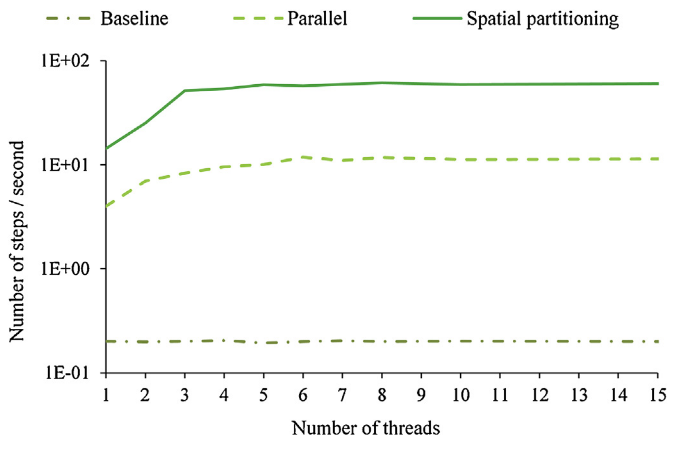

Fig. 12. Performance of the proposed approaches over threads.

4.3. Effect of number of threads over the proposed approaches

During simulation, the time spent in creating and positioning the agents is directly dependent on the number of threads initially configured by the user.

As Fig. 12 shows, the performance of the multi-thread and partitioning approaches increases when a larger number of threads are available. In the case of the parallel alternative (dashed line), the performance is improved because more agents can be concurrently executed. In the case of the spatial partitioning approach (solid line), the performance of the simulation improves, because the environment is divided into smaller partitions, each one running at a different thread and partially autonomous of the rest. When only a single thread is used, the distributed approaches also outperform the sequential approach (dash dotted line), because the agent positioning schemes of distributed approaches are optimised and enable the execution of more simulation steps per second.

However, there is a limit to the improvement achieved by increasing the number of threads depending on the approach to be used. In the parallel approach, threads block the agents in use. Consequently, more threads imply an increased delay in thread completion, because threads will need to wait for one another to complete actions dependent on the population rather than the individual (e.g. agent repositioning or agent binding). In the spatial partitioning approach, the division of the environment in very small partitions implies that each partition will handle a limited subset of agents and the number of transitions between partitions will likely escalate. Therefore, the sequential execution time spent in agent transition operations will increase.

Further details on these experiments can be found in Supplementary material S4.

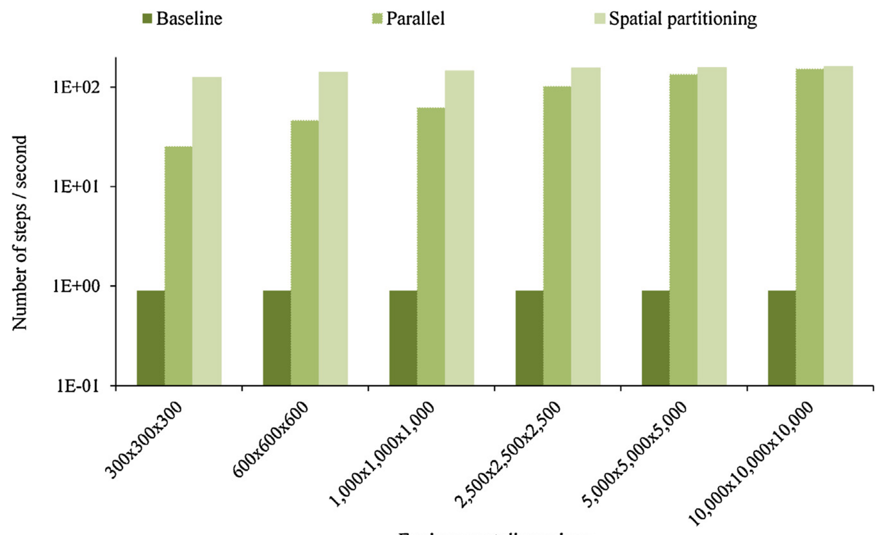

Fig. 13. Performance of the proposed approaches over environment dimensions.

4.4. Effect of environment dimension over the proposed approaches

The definition of the environment is established based on real measurements (e.g. the volume of a bacterial cytoplasm or the volume of a bioreactor). Therefore, it is important to analyse the impact that dimensions have over the performance of the proposed approaches in order to assess which one is preferable in a given scenario. For this purpose, we considered three-dimensional continuous cubic environments of varying size.

Typically, the performance of distributed simulations is better in larger environments because the shared and partition-based position maps (structures of the parallel and spatial partitioning approaches, respectively) are able to improve the time spent in positioning the agents (Fig. 13). As the environment expands, the number of available positions on the X-axis also grows and therefore, agents tend to be more dispersed and there is a smaller probability of position conflict.

Most notably, in larger environments, the simulation rate of the distributed approaches is quite similar, because the spatial partitioning position map is less efficient when agents are more dispersed in the environment (fewer operations can be handled within partition). In turn, the baseline approach shows no change in performance, because its neighbour search algorithm screens all the agents for every single operation, regardless of the size of the environment. Further details on these experiments can be found in Supplementary material S5.

5. Conclusions and further work

Arguably, three-dimensional continuous simulations of the interplay of individual molecules are among the most demanding *in silico* simulations. The modelling of realistic cellular scenarios can easily encompass thousands of agents and the time scale to monitor the interplay can easily go from seconds to nanoseconds. Existing high performance individual particle approaches majorly rely on specialised computing architectures and highly specialised programming. Therefore, these approaches have a restricted use from the general Life Science community. In contrast, our work explores less specialised strategies that may be broadly implemented in the computers of Life Sciences research centre facilities. Specifically, the main goal of our work was to devise more amicable approaches to three-dimensional continuous molecular simulation that could be put in practice in event-driven ABM frameworks and machines with moderate computing power.

The simulation of a simple enzymatic model was used as basis of validation and comparison. The three proposed approaches produced reproducible results coherent with main biophysical and biochemical laws and enabled three-dimensional spatial analysis, something not fully supported by existing tools. Regarding simulation performance, the size of the agent population is a critical factor: our parallel and spatial partitioning approaches showed improved performance in large size populations whereas the sequential approach performed better in small to medium size populations (below 10,000 individuals).

Overall, our rationale is that if simulation is closer to the Life Sciences community, the contributions and feedback will likely be more active and productive. Especially, common biophysical and biochemical assumptions taken by more traditional biological simulation approaches may be revised at the light of novel experimental observations. So, providing tools that are at the reach of most biologists is of major interest to promote a more generalised execution of these simulations, helping improve the tools while delivering new insights into key cellular processes. In this context, the new approaches that we present are a useful and new contribution to high performance three-dimensional molecular simulation and, hopefully, they can assist the work of a large number of researchers.

Acknowledgments

This work was partially funded by the [14VI05] Contract-Programme from the University of Vigo and the Agrupamento INBIOMED from DXPCTSUG-FEDER unha maneira de facer Europa (2012/273). The research leading to these results has received funding from the European Union's Seventh Framework Programme FP7/REGPOT-2012-2013.1 under grant agreement n°316265, BIOCAPS. This document reflects only the author's views and the European Union is not liable for any use that may be made of the information contained herein.

Appendix A. Supplementary data

Supplementary data associated with this article can be found, in the online version, at http://dx.doi.org/10.1016/j.jmgm.2016.06.001.

References

[1] G. An, S. Kulkarni, An agent-based modeling framework linking inflammation and cancer using evolutionary principles: description of a generative hierarchy for the hallmarks of cancer and developing a bridge between mechanism and epidemiological data, Math. Biosci. (2014), http://dx.doi.org/10.1016/j.mbs.2014.07.009.

[2] S.S. Andrews, N.J. Addy, R. Brent, A.P. Arkin, Detailed simulations of cell biology with Smoldyn 2.1, PLoS Comput. Biol. 6 (2010) e1000705, http://dx.doi.org/10.1371/journal.pcbi.1000705.

[3] A. Apte, R. Senger, S.S. Fong, Designing novel cellulase systems through agent-based modeling and global sensitivity analysis, Bioengineered (2014) 5.

[4] A.O. Barden, A.S. Goler, S.C. Humphreys, S. Tabatabaei, M. Lochner, M.-D. Ruepp, T. Jack, J. Simonin, A.J. Thompson, J.P. Jones, J. a. Brozik, Tracking individual membrane proteins and their biochemistry: the power of direct observation, Neuropharmacology 1-9 (2015), http://dx.doi.org/10.1016/j.neuropharm.2015.05.003.

[5] K. Bettenbrock, H. Bai, M. Ederer, J. Green, K.J. Hellingwerf, M. Holcombe, S. Kunz, M.D. Rolfe, G. Sanguinetti, O. Sawodny, P. Sharma, S. Steinsiek, R.K. Poole, Towards a systems level understanding of the oxygen response of Escherichia coli, Adv. Microb. Physiol. 64 (2014) 65–114, http://dx.doi.org/10.1016/B978-0-12-800143-1.00002-6.

[6] M.J. Byrne, M.N. Waxham, Y. Kubota, Cellular dynamic simulator: an event driven molecular simulation environment for cellular physiology, Neuroinformatics 8 (2010) 63–82, http://dx.doi.org/10.1007/s12021-010-9066-x.

[7] N. Cannata, F. Corradini, E. Merelli, L. Tesei, Agent-based models of cellular systems, Comput. Toxicol. Methods Mol. Biol. 930 (2013) 399–426, http://dx.doi.org/10.1007/978-1-62703-059-5.

[8] S.C. Choe, A. Hamacher-Brady, N.R. Brady, Autophagy capacity and sub-mitochondrial heterogeneity shape Bnip3-induced mitophagy regulation of apoptosis, Cell Commun. Signal. 13 (2015) 37, http://dx.doi.org/10.1186/s12964-015-0115-9.

[9] N. Collier, M. North, Parallel agent-based simulation with repast for high performance computing, Simulation 89 (2013) 1215–1235, http://dx.doi.org/10.1177/0037549712462620.

[10] R. Conte, M. Paolucci, On agent-based modeling and computational social science, Front. Psychol. 5 (2014) 668, http://dx.doi.org/10.3389/fpsyg.2014.00668.

[11] G. Cordasco, R. De Chiara, A. Mancuso, D. Mazzeo, V. Scarano, C. Spagnuolo, Bringing together efficiency and effectiveness in distributed simulations: the experience with D-Mason, Simulation 89 (2013) 1236–1253, http://dx.doi.org/10.1177/0037549713489594.

[12] A.E. Curtin, L. Zhou, An agent-based model of the response to angioplasty and bare-metal stent deployment in an atherosclerotic blood vessel, PLoS One 9 (2014) e94411, http://dx.doi.org/10.1371/journal.pone.0094411.

[13] J.O. Dada, P. Mendes, Multi-scale modelling and simulation in systems biology, Integr. Biol. (Camb) 3 (2011) 86–96, http://dx.doi.org/10.1039/c0ib00075b.

[14] M. Feig, Y. Sugita, Reaching new levels of realism in modeling biological macromolecules in cellular environments, J. Mol. Graphic Modell. 45 (2013) 144–156, http://dx.doi.org/10.1016/j.jmgm.2013.08.017.

[15] G. Foffi, A. Pastore, F. Piazza, P.A. Temussi, Macromolecular crowding: chemistry and physics meet biology (Ascona, Switzerland, 10–14 June 2012), Phys. Biol. 10 (2013) 040301, http://dx.doi.org/10.1088/1478-3975/10/4/040301.

[16] N.D. Fredrick, J.A. Berges, B.S. Twining, D. Nuñez-Milland, F.L. Hellweger, Use of agent-based modeling to explore the mechanisms of intracellular phosphorus heterogeneity in cultured phytoplankton, Appl. Environ. Microbiol. 79 (2013) 4359–4368, http://dx.doi.org/10.1128/AEM.00487-13.

[17] P.P. González Pérez, A. Omicini, M. Sbaraglia, A biochemically-inspired coordination-based model for simulating intracellular signalling pathways, J. Simul. 7 (2013) 216–226, http://dx.doi.org/10.1057/jos.2012.28.

[18] P.P. González, M. Cárdenas, D. Camacho, A. Franyuti, O. Rosas, J. Lagúnez-Otero, Cellulat: an agent-based intracellular signalling model, Biosystems 68 (2003) 171–185, http://dx.doi.org/10.1016/S0303-2647(02)00094-1.

[19] H. Iba, Agent-Based Modeling and Simulation with Swarm, 1st edition, Chapman & Hall/CRC Studies, 2013.

[20] S. Kang, S. Kahan, J. McDermott, N. Flann, I. Shmulevich, Biocellion: accelerating computer simulation of multicellular biological system models, Bioinformatics (2014) 1–9, http://dx.doi.org/10.1093/bioinformatics/btu498.

[21] S. Kang, S. Kahan, B. Momeni, Simulating microbial community patterning using Biocellion, Methods Mol. Biol. 1151 (2014) 233–253, http://dx.doi.org/10.1007/978-1-4939-0554-6_16.

[22] H. Kaul, Y. Ventikos, Investigating biocomplexity through the agent-based paradigm, Brief. Bioinform. 16 (2015) 137–152, http://dx.doi.org/10.1093/bib/bbt077.

[23] S. Kazachenko, M. Giovinazzo, K.W. Hall, N.M. Cann, Algorithms for GPU-based molecular dynamics simulations of complex fluids: applications to water, mixtures, and liquid crystals, J. Comput. Chem. (2015), http://dx.doi.org/10.1002/jcc.24000.

[24] M.T. Klann, A. Lapin, M. Reuss, Agent-based simulation of reactions in the crowded and structured intracellular environment: influence of mobility and location of the reactants, BMC Syst. Biol. 5 (2011) 71, http://dx.doi.org/10.1186/1752-0509-5-71.

[25] K. Kravari, N. Bassiliades, A survey of agent platforms, J. Artif. Soc. Soc. Simul. 18 (2015), http://dx.doi.org/10.18564/jasss.2661.

[26] J.E. Ladbury, S.T. Arold, Noise in cellular signaling pathways: causes and effects, Trends Biochem. Sci. 37 (2012) 173–178, http://dx.doi.org/10.1016/j.tibs.2012.01.001.

[27] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, G. Balan, MASON: a multiagent simulation environment, Simul. Trans. Soc. Model. Simul. Int. 82 (2005) 517–527, http://dx.doi.org/10.1177/0037549705058073.

[28] S.L. Lytinen, S.F. Railsback, The evolution of agent-based simulation platforms: a review of NetLogo 5.0 and ReLogo, Proceedings of the Fourth International Symposium on Agent-Based Modeling and Simulation (21 st European Meeting on Cybernetics and Systems Research) (2012).

[29] A. Masoudi-Nejad, G. Bidkhori, S. Hosseini Ashtiani, A. Najafi, J.H. Bozorgmehr, E. Wang, Cancer systems biology and modeling: microscopic scale and multiscale approaches, Semin. Cancer Biol. 30 (2015) 60–69, http://dx.doi.org/10.1016/j.semcancer.2014.03.003.

[30] S.R. McGuffee, A.H. Elcock, Diffusion, crowding & protein stability in a dynamic molecular model of the bacterial cytoplasm, PLoS Comput. Biol. 6 (2010) e1000694, http://dx.doi.org/10.1371/journal.pcbi.1000694.

[31] E. Merelli, G. Armano, N. Cannata, F. Corradini, M. d'Inverno, A. Doms, P. Lord, A. Martin, L. Milanesi, S. Möller, M. Schroeder, M. Luck, Agents in bioinformatics, computational and systems biology, Brief. Bioinf. 8 (2007) 45–59, http://dx.doi.org/10.1093/bib/bbl014.

[32] I. Millington, Game Physics Engine Development: How to Build a Robust Commercial-Grade Physics Engine for Your Game, CRC Press, 2010.

[33] A. Mogilner, D. Odde, Modeling cellular processes in 3D, Trends Cell Biol. 21 (2011) 692–700, http://dx.doi.org/10.1016/j.tcb.2011.09.007.

[34] S. Montagna, A. Ricci, A. Omicini, A&A for modelling and engineering simulations in systems biology, Int. J. Agent-Oriented Softw. Eng. 2 (2008) 222–245, http://dx.doi.org/10.1504/IJAOSE.2008.017316.

[35] M.J. North, N.T. Collier, J.R. Vos, Experiences creating three implementations of the repast agent modeling toolkit, ACM Trans. Model. Comput. Simul. 16 (2006) 1–25, http://dx.doi.org/10.1145/1122012.1122013.

[36] M.J. North, N.T. Collier, J. Ozik, E.R. Tatara, C.M. Macal, M. Bragen, P. Sydelko, Complex adaptive systems modeling with Repast Simphony, Complex Adapt. Syst. Model. 1 (2013) 3, http://dx.doi.org/10.1186/2194-3206-1-3.

[37] G. Palmer, Physics for Game Programmers, Apress, 2005, 978-1-59059-472-8.

[38] M. Pogson, R. Smallwood, E. Qwarnstrom, M. Holcombe, Formal agent-based modelling of intracellular chemical interactions, Biosystems 85 (2006) 37–45, http://dx.doi.org/10.1016/j.biosystems.2006.02.004.

[39] S.F. Railsback, S.L. Lytinen, S.K. Jackson, Agent-based simulation platforms: review and development recommendations, Simulation 82 (2006) 609–623, http://dx.doi.org/10.1177/0037549706073695.

[40] P. Richmond, D. Walker, S. Coakley, D. Romano, High performance cellular level agent-based simulation with FLAME for the GPU, Brief. Bioinf. 11 (2010) 334–347, http://dx.doi.org/10.1093/bib/bbp073.

[41] S. Riniker, J.R. Allison, W.F. van Gunsteren, On developing coarse-grained models for biomolecular simulation: a review, Phys. Chem. Chem. Phys. 14 (2012) 12423, http://dx.doi.org/10.1039/c2cp40934h.

[42] J. Schöneberg, F. Noé, ReaDDy-a software for particle-based reaction-diffusion dynamics in crowded cellular environments, PLoS One 8 (2013) e74261, http://dx.doi.org/10.1371/journal.pone.0074261.

[43] J. Schöneberg, A. Ullrich, F. Noé, Simulation tools for particle-based reaction-diffusion dynamics in continuous space, BMC Biophys. 7 (2014) 11, http://dx.doi.org/10.1186/s13628-014-0011-5.

[44] E. Sklar, NetLogo, a multi-agent simulation environment, Artif. Life 13 (2007) 303–311, http://dx.doi.org/10.1162/artl.2007.13.3.303.

[45] J.R. Stern, A.D. Olivas, V. Valuckaite, O. Zaborina, J.C. Alverdy, G. An, Agent-based model of epithelial host-pathogen interactions in anastomotic leak, J. Surg. Res. 184 (2013) 730–738, http://dx.doi.org/10.1016/j.jss.2012.12.009.

[46] S.M. Stiegelmeyer, M.C. Giddings, Agent-based modeling of competence phenotype switching in Bacillus subtilis, Theor. Biol. Med. Model. 10 (2013) 23, http://dx.doi.org/10.1186/1742-4682-10-23.

[47] R. Tobias, C. Hofmann, Evaluation of free Java-libraries for social-scientific agent based simulation, J. Artif. Soc. Soc. Simul. (2004) 7.

[48] R. Wade, J. McCammon, Biological Diffusion and Brownian Dynamics, Spec. Themat. Ser. Ina., Vol. 4 BMC Biophys.

[49] D.C. Walker, J. Southgate, G. Hill, M. Holcombe, D.R. Hose, S.M. Wood, S. Mac Neil, R.H. Smallwood, The epitheliome: agent-based modelling of the social behaviour of cells, Biosystems 76 (2004) 89–100, http://dx.doi.org/10.1016/j.biosystems.2004.05.025.

[50] C.P. Whitman, B.A. Aird, W.R. Gillespie, N.J. Stolowich, Chemical and enzymic ketonization of 2-hydroxymuconate, a conjugated enol, J. Am. Chem. Soc. 113 (1991) 3154–3162, http://dx.doi.org/10.1021/ja00008a052.

[51] J. Yu, Coordination and control inside simple biomolecular machines, Adv. Exp. Med. Biol. 805 (2014) 353–384, http://dx.doi.org/10.1007/978-3-319-02970-2_15.