Chapter 16

Simulating Microbial Community Patterning Using *Biocellion*

Seunghwa Kang, Simon Kahan, and Babak Momeni

Abstract

Mathematical modeling and computer simulation are important tools for understanding complex interactions between cells and their biotic and abiotic environment: similarities and differences between modeled and observed behavior provide the basis for hypothesis formation. Momeni et al. (Elife 2:e00230, 2013) investigated pattern formation in communities of yeast strains engaging in different types of ecological interactions, comparing the predictions of mathematical modeling, and simulation to actual patterns observed in wet-lab experiments. However, simulations of millions of cells in a three-dimensional community are extremely time consuming. One simulation run in MATLAB may take a week or longer, inhibiting exploration of the vast space of parameter combinations and assumptions. Improving the speed, scale, and accuracy of such simulations facilitates hypothesis formation and expedites discovery. *Biocellion* is a high-performance software framework for accelerating discrete agent-based simulation of biological systems with millions to trillions of cells. Simulations of comparable scale and accuracy to those taking a week of computer time using MATLAB require just hours using *Biocellion* on a multicore workstation. *Biocellion* further accelerates large scale, high resolution simulations using cluster computers by partitioning the work to run on multiple compute nodes. *Biocellion* targets computational biologists who have mathematical modeling backgrounds and basic C++ programming skills. This chapter describes the necessary steps to adapt the original Momeni et al.'s model to the *Biocellion* framework as a case study.

Key words Discrete agent-based modeling, Partial differential equation, Adaptive mesh refinement, High-performance computing, Cell system simulation

1 Introduction

Discrete agent-based modeling maps a multicellular biological system to a collection of discrete agents. Discrete agent-based modeling has been widely used to model various biological systems [1, 3–5, 7]. Momeni et al. [5] studied spatial patterning in a community of yeast strains engaging in metabolic interactions through a combination of discrete agent-based mathematical modeling and wet-lab experiments.

Finding solutions using mathematical models often requires implementation of these models as a computer program.

Producing a high-performance implementation that anticipates and accommodates easy revisions and refinements of a model as it evolves is a time-consuming task. Computational biologists often favor low programming effort over efficiency, flexibility, and even accuracy. Therefore, simulations tend to run much longer than performance-optimized code. Even incremental model updates can require significant code revision. For example, Momeni et al.'s model [5] partitions the simulation domain into fixed size cubic boxes and maps each cell to a single box. Once a cell divides, the daughter cell occupies one of the nearest neighboring boxes, instantly pushing surrounding cells outward to free up the space. This approximation of cell growth and shoving may be sufficient for the studied problem, but it may not work well for more complex problems—e.g., this approach cannot model a cell division that produces two cells differing in size.

Parallel computers ranging from multicore PCs to leadership class supercomputers provide significantly larger computing capacity than a single compute core. This computing capacity can address the computational challenges in simulating complex models when harnessed by *simulation software that serves widely varying multicellular biological system modeling requirements and runs efficiently on parallel computers.* However, implementing such software is difficult and time consuming. Even if computational biologists have access to cluster computers, without parallel computing, the advantage is limited to running multiple simulations in parallel—one simulation per compute core—resulting in slow turn-around time. In contrast, with efficient parallel software, a single simulation run can be partitioned across multiple computing cores, dramatically reducing the turn-around time.

Biocellion is a high-performance software framework that enables computational biologists without parallel computing expertise to exploit the power of parallel computers with only moderate programming effort. We briefly describe Momeni et al.'s model [5] and *Biocellion* [6] and illustrate the necessary steps to adapt Momeni et al.'s model to *Biocellion* as a case study.

1.1 Yeast Patterning Model Description

Momeni et al. [5] implemented a mathematical model with a range of parameters corresponding to different yeast strains engaging in various interactions. We consider only one instance in this article. Modification for other strains and interaction types is straightforward.

Two strains of yeast grow on top of a 24 mm thick agarose cylinder. One strain consumes lysine and also secretes adenine at a constant rate (say the R strain). The other strain consumes adenine and releases lysine on death (say the G strain). The two strains grow cooperatively and Momeni et al. demonstrated that strong cooperation promotes intermixing of the two strains in a three-dimensional community. We list model specifics necessary to adapt the model to *Biocellion*.

1. Yeast cells grow on top of a 24,000 μm thick agarose cylinder. Partial differential equations (PDE) are commonly used to model spatio-temporal variation of molecular concentrations in the extracellular space. We adopt this approach, and PDEs model adenine and lysine concentration changes in the agarose cylinder and the yeast cell community. Because the initial spatial distribution of the two yeast strains is uniform, a small fraction of the plate area in wet-lab experiment is already representative of patterns observed for the entire plate. Thus, only a fraction of the plate area in wet-lab experiments is considered in simulation. In Momeni et al.'s work [5], simulation domain height is set to include the entire agarose cylinder in the simulation area plus up to 300 μm above the agarose cylinder where yeast cells grow.
2. Periodic boundary conditions (both for PDEs and cell movements) are assumed in the *x* and *y* directions. A zero-flux boundary condition is applied at the bottom end of the agarose cylinder and at the top of the simulation domain—molecules and cells cannot cross the top and bottom planes of the simulation domain.
3. We map a cell to a sphere (instead of a fixed size cubic box in the original model) using *Biocellion*. The maximum cell diameter is 5 μm. The maximum cell volume is $\frac{4 \times \pi \times 2.5^3}{3} \mu m^3$. Yeast cells push against other cells when they are packed together.
4. We increase the volume of a sphere to model cell growth. The volume of an *R* strain cell after consuming a certain amount of lysine (say Δlysine) is $V_0 \times \left(1 + \frac{\Delta \text{lysine}}{\alpha_L}\right)$, where V_0 is the minimum cell volume (the volume of a cell right after cell division or one half of the maximum cell volume). α_L, the amount of lysine required to produce a daughter cell, is 2 fmol. Similarly, the volume of a *G* strain cell after consuming a certain amount of adenine (say Δadenine) is $V_0 \times \left(1 + \frac{\Delta \text{adenine}}{\alpha_A}\right)$. α_A, the amount of adenine required to produce a daughter cell, is 1 fmol. When a cell grows above the maximum cell volume, the cell divides into two equal-volume cells.
5. For an *R* strain cell, lysine uptake rate is $v_L = \frac{\alpha_L}{T_L} \times \frac{\phi_L}{K_L + \phi_L}$.

   For a *G* strain cell, adenine uptake rate is $v_A = \frac{\alpha_A}{T_A} \times \frac{\phi_A}{K_A + \phi_A}$.

The minimal population doubling times for lysine-requiring and adenine-requiring cells (T_L and T_A, respectively) are 1.76 and 1.98 h, respectively. The *Monod's constant* K_L, i.e., the concentration of lysine at which lysine-requiring cells grow at their half maximal growth rate is 1 μM. The Monad's constant K_A for adenine requiring cells is 0.1 μM. ϕ_L and ϕ_A are lysine and adenine concentrations in the extracellular space, respectively. An *R* strain cell secretes 0.08 fmol of adenine per hour, and a *G* strain cell releases 12 fmol of lysine on death.

6. The death rates are 0.054 h^{-1} and 0.018 h^{-1} for an *R* strain cell and a *G* strain cell, respectively.
7. Cells are randomly distributed on the agarose cylinder surface at the beginning of the simulation. Initial cell volume is set to a random value between one half of the maximum cell volume and the maximum cell volume to represent the range between a new-born daughter and a fully grown cell. The initial cell density is 500 cells per mm^2.
8. Initial lysine and adenine concentrations are set to zero.
9. Diffusion coefficients are 300 $\mu m^2/s$ in the agarose cylinder and 20 $\mu m^2/s$ inside a yeast colony for both lysine and adenine according to experimental measurements. Diffusion coefficients for grid boxes containing yeast cells are scaled down based on the total volume of the cells in a grid box—a grid box with low cell volume (a box in the colony-air boundary) has smaller diffusion coefficients than a box with high cell volume as implemented in the original model.

1.2 Biocellion Overview

Biocellion's design goal is to accelerate a wide range of discrete agent-based mathematical models of multicellular biological systems. This is challenging, because mathematical models of biological systems vary significantly. *Biocellion*'s approach is to separate model specifics from common computational and parallel programming challenges. *Biocellion* asks users to provide model specifics, and *Biocellion* handles the remaining computational and programming challenges. Model specifics are expressed by the developer through modification of a library of C++ functions that comprise *Biocellion*'s Application Programming Interface (API). The model library links to the *Biocellion* core framework at runtime. *Biocellion* output files can be visualized using Paraview (http://www.paraview.org).

Biocellion has three computational modules to simulate (1) individual discrete agent behavior, (2) direct physico-mechanical interactions between discrete agents, and (3) changes in the extracellular environment. *Biocellion* imposes a grid on the simulation domain to represent the state of the extracellular environment. In Momeni et al.'s model [5], cells reside on top of the 24 mm thick agarose cylinder, and the region where yeast cells grow is a small fraction of the entire simulation domain. The agarose cylinder is relevant only in tracking molecular concentrations in the model. Maintaining data structures for all three computational modules for the entire agarose cylinder can waste a significant amount of computing and memory. *Biocellion* imposes two different types of grids to different parts of the simulation domain to avoid such waste. *Biocellion* imposes an interface grid on a region where all three computational modules are executed, and computational modules communicate through this interface grid. *Biocellion* imposes a coarser PDE buffer grid on the region relevant only in solving PDEs—e.g., tracking nutrient concentrations in the agarose cylinder.

Biocellion decomposes the simulation domain into multiple partitions—users set the partition size. Users can impose either an interface grid or a PDE buffer grid for each partition. When users wish to run *Biocellion* on a cluster computer with multiple compute nodes (each node has multiple compute cores), *Biocellion* creates multiple compute processes, and every compute process works on a different set of partitions; note that a single process can exploit multiple compute cores in a single compute node to work on a single partition. Separate output files are created for different partitions.

Biocellion supports adaptive mesh refinement (AMR) to solve PDEs. AMR generates multiple levels of grids with different grid spacings based on the spatial resolution requirements of different simulation domain subregions. *Biocellion* asks users to set the number of AMR levels and the refinement ratio between two consecutive AMR levels—if the refinement ratio is set to 4, the coarser level grid spacing is four times larger than the finer level grid spacing. The finest grid spacing coincides with the interface grid spacing—*Biocellion* users set the interface grid spacing. The PDE buffer grid spacing equals the coarsest grid spacing in the AMR hierarchy. Users tag interface grid boxes with the desired AMR level. PDE buffer grid boxes are automatically tagged with the coarsest AMR level. *Biocellion* generates an AMR hierarchy (which is used to solve PDEs) based on this information. Note that the generated AMR hierarchy can have more fine boxes than the user input to improve efficiency (processing a large number of small boxes is inefficient) and guarantee correctness (coarsening a fine grid first and refining the coarsened fine grid should produce the original fine grid, or see the proper nesting condition in ref. [2]).

Figure 1 depicts the simulation domain (left) and the generated AMR hierarchy (right) in our experiment assuming 40 μm grid spacing, two AMR levels, and the refinement ratio of 4—we simulate with 5, 20, and 40 μm grid spacings.

Multicellular biological system simulation combines multiple biological processes such as cell movement, diffusion of molecules, and cell metabolic rate change. Different biological processes have

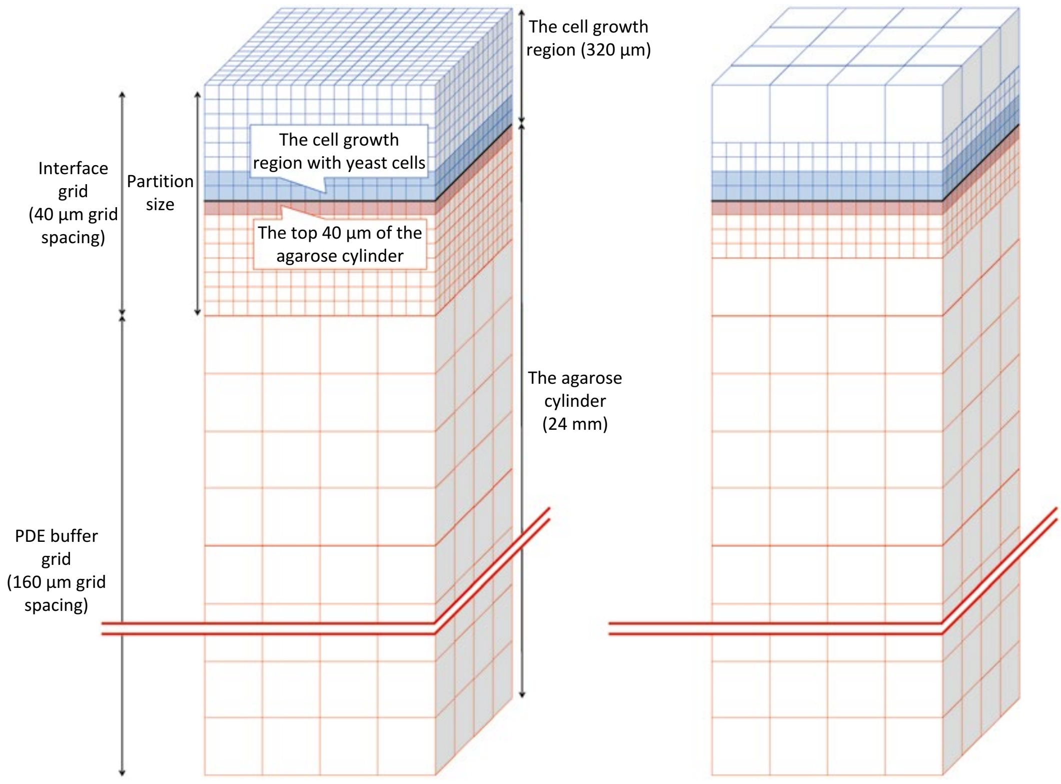
The cell growth region (320 μm)
The cell growth region with yeast cells
Interface grid (40 μm grid spacing)
Partition size
The top 40 μm of the agarose cylinder
The agarose cylinder (24 mm)
PDE buffer grid (160 μm grid spacing)

Fig. 1 *Biocellion* imposes two different types of grids to the simulation domain (*left*). An interface grid is imposed on the partition covering the cell growth region to simulate both cells and the environment. The remaining partitions in the agarose cylinder region are set as PDE buffer—these partitions are relevant only in solving PDEs to track lysine and adenine concentrations. Users tag interface gird boxes with the desired AMR level. We tag the interface grid boxes containing yeast cells (the *light blue boxes*) and the interface grid boxes at the top 40 μm of the agarose cylinder (the *light red boxes*) with the finer AMR level. The remaining interface grid boxes are tagged with the coarser level. *Biocellion* generates an AMR hierarchy (*right*) based on this information. Note that the generated AMR hierarchy has more fine boxes than the user input to satisfy the proper nesting condition [7]

different time step requirements to simulate the processes with sufficient accuracy. For example, simulating diffusion of molecules (by solving PDEs) often requires a significantly smaller time step size than the time step size required to simulate cell movement. To accommodate multiple time step size requirements in multicellular biological system simulation, *Biocellion* uses multiple time step sizes to simulate different computational modules and to communicate across the modules. The *baseline time step* is the largest time step used to simulate direct physico–mechanical interactions and discrete agent birth, death, and movement. The module computing direct physico–mechanical interactions communicates with the other two modules once per baseline time step. Discrete agent states and the state of the extracellular environment affect each other. For example, cell metabolic rate change affects the production and consumption rates of extracellular molecules, and this drives molecular concentration changes in the extracellular space. These two modules can be coupled more tightly by splitting a single *baseline time step* into multiple *state-and-grid time steps*. Variables associated with the grid imposed on the extracellular space can be updated either by model specific rules or by solving PDEs. Users can update the variables by model specific rules at the beginning and at the end of each *state-and-grid time step*. A single *state-and-grid time step* can be further partitioned to smaller *PDE time steps* to advance PDEs.

1.3 Porting Overview

The original yeast patterning model [5] partitions the simulation domain on top of the agarose cylinder into a set of fixed size cubic boxes (a box width is 5 μm), and a cell takes a single box. If a cell divides, the new cell tries to occupy one of the nearest neighboring boxes in the same *z* plane, if there is an empty box within the *confinement neighborhood* of 5-cell radius. The existence of confinement neighborhood was observed experimentally. If there is no empty box within the confinement neighborhood in the same *z* plane, the new cell occupies the box right on top of the mother cell box, and all the other cell boxes on top of the mother cell box are pushed upward.

Using *Biocellion*, we represent each cell by a sphere (*Biocellion* allows users to map a discrete agent to a different shape). A sphere can be located anywhere in the simulation domain above the agarose cylinder, and its radius changes to model cell growth. When a cell grows just enough to overlap with another, the model immediately introduces a force to push all spheres apart, thus modeling cell shoving in packed regions.

Figure 2 shows that the concentration of adenine changes smoothly in the simulation domain except for the cell growth region and the top 40 μm part of the agarose cylinder—adenine concentration changes smoothly even just 40 μm below the agarose cylinder top surface. We set the partitions at the agarose cylinder region (except for the top 40 μm part) as PDE buffer. *Biocellion* supports AMR which applies different grid resolutions to different parts of the simulation domain. In generating an AMR hierarchy, the region occupied by yeast cells and the top 40 μm of the agarose cylinder are tagged with the finest grid spacing, which is equal to the interface grid spacing. A coarser grid is imposed on the air region and the bottom part of the agarose cylinder.

We set the *baseline time step size* to 30 s and split a single *baseline time step* to 30 *state-and-grid time steps* to tightly couple cell metabolic rate change and nutrient concentration change in the extracellular space. *PDE time step* sizes to advance PDEs updating lysine and adenine concentrations are set identical to the *state-and-grid time step* size.

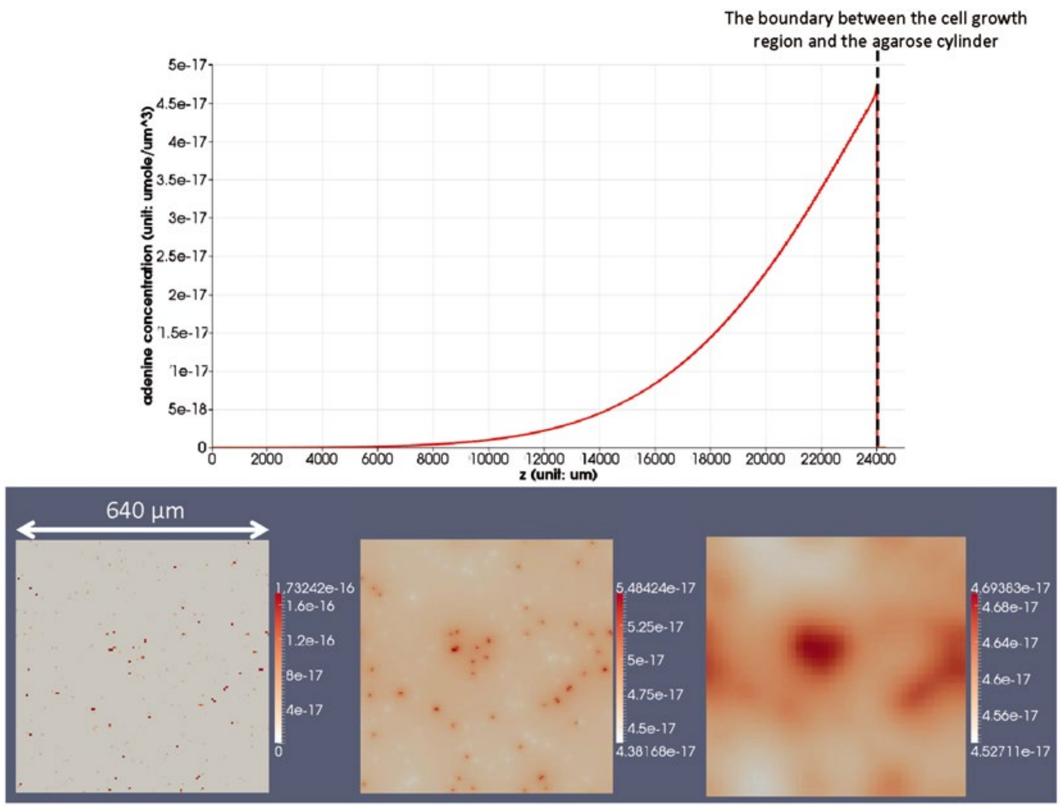

Fig. 2 Adenine concentrations in the simulation domain (unit: μmol/μm^3, 1 μM = 10^{-15}μmol/μm^3). The *top figure* depicts the adenine concentration along the *z*-axis (passing the center of the simulation domain), with 0 and 24,000 being the bottom and the top of the agarose cylinder, respectively. The *bottom figures* show the adenine concentration at the *z* normal planes right on top of the agarose cylinder (*right*), right below the agarose cylinder top surface (*center*), and 40 μm below the agarose cylinder top surface (*right*), respectively. The adenine concentration has higher spatial variation near the agarose cylinder top surface. The spatial variation of the adenine concentration is significantly lower even just 40 μm below the top surface

We define four model specific variables (say rhs_{lysine}, $rhs_{adenine}$, $U_{scale,lysine}$, and $U_{scale,adenine}$) for each grid box in the interface grid. rhs_{lysine}, $rhs_{adenine}$ store the sum of the production and consumption rates of lysine and adenine, respectively. Lysine and adenine consumption rates are proportional to $\frac{\phi}{\phi + K}$, where ϕ is lysine or adenine concentration in the extracellular space and K (the concentration of metabolite at which half maximal consumption rate is achieved) is 1.0 and 0.1 μM for lysine and adeneine, respectively. $U_{scale,lysine}$, and $U_{scale,adenine}$ store the $\frac{\phi}{\phi + K}$ values for lysine and adenine, respectively. We want to limit the total amount of lysine or adenine consumed by cells in a grid box to be lower than the amount of lysine or adenine in the box plus an estimation of the amount of lysine or adenine diffuse into the box within a single *state-and-grid time step*. This prevents ϕ from becoming negative (especially when ϕ is small) without using a tiny time step size. This approach is accurate as long as our estimation of the diffusion rate is accurate. We use the explicit Euler method to estimate the amount of diffusion, which gives a reasonably accurate estimation for our choice of the *state-and-grid time step* size (1 s)—the molecular concentration gradient does not change significantly within 1 s. We scale $\frac{\phi}{\phi + K}$ to limit nutrient consumption. If the sum of rhs_{lysine} (or $rhs_{adenine}$) and the estimated diffusion rate multiplied by the *state-and-grid time step* size exceeds the lysine (or adenine) concentration of the box, we reduce the consumption rate and scale down $U_{scale,lysine}$ (or $U_{scale,adenine}$), so the net decrease of the lysine (or adenine) concentration based on the production, consumption, and estimated diffusion rates does not exceed the lysine concentration of the grid box. Model routines setting PDE parameters and model routines updating individual cell states can access these values to set nutrient uptake rates—this is necessary to assure that the total amount of nutrients consumed in solving PDEs coincide with the amount consumed by cells in updating individual cell states. We save rhs_{lysine} and $rhs_{adenine}$ to avoid computing the rates again when setting PDE parameters.

A *G* strain cell releases lysine on death. A relatively large amount of lysine is released in a short amount of time, and this forms a steep concentration gradient followed by a rapid gradient change due to diffusion. Accurately computing this transient gradient change requires a small time step size. However, cells react to concentration changes only gradually, so accurately computing the transient gradient change has little impact on simulation output. We have decided to spread released lysine to six neighboring boxes in the $\pm x$, y, and z directions to lower the initial concentration gradient. We implement this by updating the rhs_{lysine} variable of a neighboring grid boxes.

Figure 3 shows a simulation output. Simulation time is highly dependent on grid resolution and with 40 μm interface grid spacing (comparable to 50 *mum* grid spacing used in the Momeni et al.'s work [5]) to simulate 500 h of cell growth, a single simulation run takes 6.5 h on a workstation with a single 6 core microprocessor (Intel X5650 2.67 GHz). *Biocellion* also accelerates larger higher resolution simulations using multiple compute nodes.

2 Materials

Biocellion runs on multicore PCs, workstations, cluster computers, cloud computers, and supercomputers. This article pertains to multicore PCs. Running on different systems does not require model code changes. The current version of *Biocellion* runs only on ×86 compatible systems (PCs with Intel or AMD microprocessors are ×86 compatible). A 64-bit Linux operating system needs to be

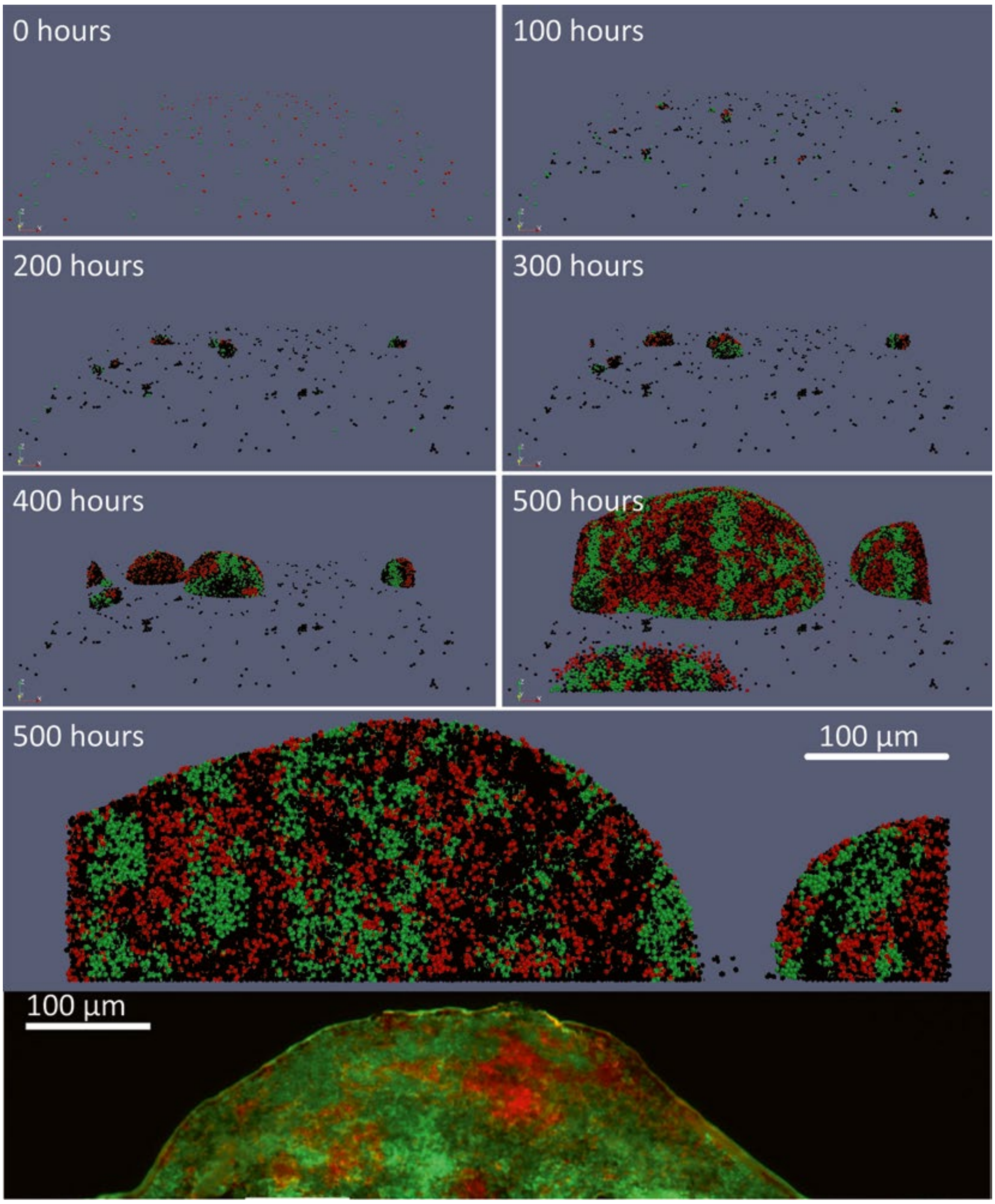

Fig. 3 Yeast cell growth (the top six figures) viewed from an oblique angle from the top, a 2D vertical cross-section of the yeast colony (*the next figure*), and a 2D vertical cross-section from the wet-lab experiment (*the bottom most figure*, reproduced from Momeni et al.'s paper [5]). *Red cells* are R strain cells, *green cells* are G strain cells, and *black cells* are dead cells

installed on the target system. Compiling *Biocellion* model code requires the GNU gcc compiler (pre-installed in most Linux systems), Intel icc compiler, or some other C++ compiler (we have tested only with gcc and icc). When compiling model code, users may set the check flag to verify their code or disable the check for higher performance (*See* **Note 1**). *Biocellion* requires the Intel Thread Building Blocks library, freely available from the thread building blocks homepage (http://threadingbuildingblocks.org).

2.1 Installing Intel Thread Building Blocks

1. Download the most recent stable version of Intel Thread Building Blocks library (version 4.2 or later is required) to the target system.
2. Unzip the downloaded tarball.
3. Update the `LD_LIBRARY_PATH` Linux environment variable to include the TBB library directory (in TBB 4.2, this is `$TBB_ROOT/lib/intel64/gcc4.1`).

2.2 Installing Biocellion

1. Unzip the *Biocellion* tarball.
2. Open Makefile.common under the *Biocellion* root directory.
3. Update `BIOCELLION_ROOT` to point to the *Biocellion* root directory.
4. Try "make" under the libmodel directory. This should compile the model library (`$BIOCELLION_ROOT/libmodel/interface/libmodel.so`).

3 Methods

Biocellion users provide model specifics by filling-in a set of C++ functions defined in five files under the `$BIOCELLION_ROOT/libmodel/model` directory: `model_routine_config.cpp`, `model_routine_agent.cpp`, `model_routine_mech_intrct.cpp`, `model_routine_grid.cpp`, and `model_routine_output.cpp`. These files include model routines to initialize the model, update discrete agent states, simulate direct physico–mechanical interactions, update the state of the extracellular space, and set simulation output, respectively. The entire model code for the Momeni et al.'s model [5] is available under `$BIOCELLION_ROOT/libmodel/model-yeast-patterning` for interested readers. Below, we present examples of how a model is specified.

3.1 Model Configuration

Model routines related to model configuration are defined in `model_routine_-config.cpp`.

1. *updateIfGridSpacing:* Set the interface grid spacing by filling the *updateIfGridSpacing* function body surrounded by/* MODEL START */and/* MODEL END */(Code 1). The interface grid spacing should be equal to or larger than the maximum direct physico-mechanical interaction distance. We map a cell to a sphere-shaped discrete agent and only consider cell shoving in evaluating short-range mechanical interactions.

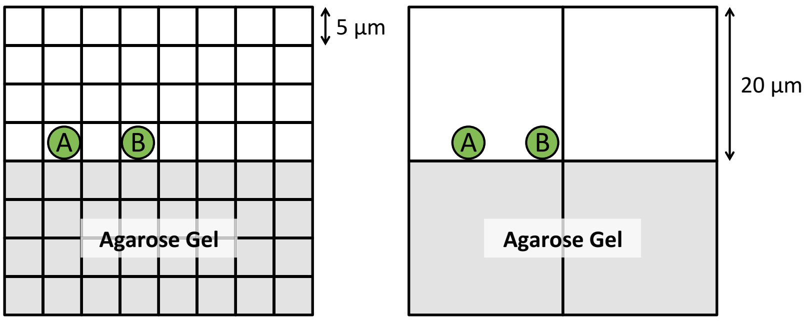

Fig. 4 Comparing 5 μm grid spacing and 20 μm grid spacing. Grid boxes outside the agarose cylinder with no cells have 0 diffusion coefficient. If cell *A* secretes a metabolite consumed by cell *B*, the secreted metabolite is delivered to cell *B* via diffusion through the agarose cylinder. However, if cell *A* and cell *B* are located in the same box (with 20 μm grid spacing), cell *B* can directly consume the molecules secreted by cell *A*

Two overlaping spheres are pushed apart to remove the overlap, and the maximum mechanical interaction distance cannot exceed the maximum cell diameter (5 μm). We start with the smallest interface grid spacing (5 μm) to minimize simulation artifacts (*see* Fig. 4 for an example). Larger values reduce execution time at potential loss of accuracy. Assuming fast diffusion, we may be able to adopt a larger grid spacing without significant loss of simulation accuracy. We also try 20 and 40 μm. Interested readers may experiment with different grid spacings to find the optimal grid spacing.

```
   void        ModelRoutine::updateIfGridSpacing
(REAL& ifGridSpacing ) {
     /* MODEL START */
     ifGridSpacing = 5.0; /* 5.0, 20.0, or 40.0,
to set the interface grid
       spacing to 5.0, 20.0, or 40.0 um */
     /* MODEL END */
     return;
   }
```

Code 1: Model routine to set the interface grid spacing.

2. *updateOptModelRoutineCallInfo*: Set the number of rounds to update variables associated with the interface grid at the beginning and at the end of a state-and-grid time step.

To set the model specific interface grid state variables properly (Subheading 1.3), *Biocellion* should be configured to invoke model routines updating interface grid state variables based on model specific rules once at the beginning of a *state-and-grid time step* and once at the end of the step (in order to reset the sum of lysine consumption and production rates, *see* **Note 2**).

3. *updateDomainBdryType*: Set domain boundary types. Periodic boundary conditions are applied in the *x* and *y* directions. A nonperiodic boundary condition is applied in the *z* direction, and cells are not allowed to pass the upper and lower end of the simulation domain in the *z* direction.
4. *updatePDEBufferBdryType*: Set the boundary type between an interface grid partition and a PDE buffer grid partition. The current version of *Biocellion* provides only one option (`PDE_BUFFER_BDRY_TYPE_HARD_WALL`), and discrete agents are not allowed to pass the boundary. This function is irrelevant to PDE boundary conditions.
5. *updateTimeStepInfo*: Update time step sizes. Set the *baseline time step size* to 30 s and split a single baseline time step to 30 *state-and-grid time steps*. Interested readers can experiment with different time step sizes.
6. *updateSyncMethod*: Update synchronization methods when a single variable is updated by multiple model routine calls (*see* **Note 2**). The only kind of short range cell–cell direct mechanical interaction we consider is cell–cell shoving, so the synchronization method for extra mechanical interactions is irrelevant. Set the synchronization method for grid variable updates to `SYNC_METHOD_DELTA` (*see* **Note 2**).
7. *updateSpAgentInfo*: Set discrete agent types. We consider three cell types (*R* and *G* strain cells and dead cells; dead cells do not grow and divide). A dead cell has one model specific variable storing the amount of lysine in the cell. This variable is used to calculate the amount of lysine to be released into the extracellular space.
8. *updatePDEInfo*: Set the grid state variables for lysine and adenine concentrations that are updated by solving PDEs. An AMR scheme is used with three levels (5 μm grid spacing) or two levels (20 μm grid spacing or 40 μm grid spacing). The finest level has the grid spacing equal to the interface grid. The size of a single PDE time step is set identical to the size of a single *state-and-grid time step*. A zero-flux boundary condition is applied in the *z* direction. Boundary conditions in the *x* and *y* directions are irrelevant, because periodic boundary conditions are imposed in *updateDomainBdryType*.
9. *updateIfGridModelVarInfo*: Set extra model specific variables associated with the interface grid. We add four variables per grid box (*see* Subheading 1.3).
10. *updateRNGInfo*: Set one random number generator to get random numbers with the uniform distribution.
11. *setPDEBuffer*: Set a partition as a PDE buffer partition if the top surface (in the *z* direction) of the partition is lower than the agarose cylinder height minus a small buffer region to

exclude the top part of the agarose cylinder (where molecular concentrations are highly localized). *setPDEBuffer* is called once per partition.

12. *setHabitable*: Set a grid box in the interface grid as a habitable box or an uninhabitable box. We set the boxes in the agarose cylinder to be uninhabitable, and cells are not allowed to move into an uninhabitable box.

3.2 Individual Agent Behavior

Model routines related to simulating individual agent behavior are defined in `model_routine_agent.cpp`.

1. *addSpAgents*: Randomly spread cells on top of the agarose cylinder to initialize the simulation. *addSpAgents* is called once per interface grid partition.
2. *updateSpAgentState*: Increase cell size based on the nutrient consumption rates. We use the scaled $\frac{\phi}{\phi + K}$ values (Subheading 1.3) to compute the nutrient consumption rates. This function is called once for every discrete agent in every *state-and-grid time step*.
3. *updateSpAgentBirthDeath*, *divideSpAgent*, and *adjustSpAgent*: Set whether a discrete agent will divide or disappear (*updateSpAgentBirthDeath*, *see* Code 2). *updateSpAgentBirthDeath* is called once for every discrete agent in every *baseline time step*. A cell divides if its size exceeds the maximum cell size. If a cell is set to divide, *divideSpAgent* is called. A cell divides into two cells in a random direction and the two resulting cells' volume is one half of the original cell's volume in the ported model. Dead cells remain in the simulation domain, so no discrete agent is set to disappear. If neither divide nor disappear is set, *adjustSpAgent* is called and this model routine updates the cell displacement based on the sum of the forces on the cell.

3.3 Physico–Mechanical Interaction Between Agents

Model routines related to simulating physico–mechanical interactions between discrete agents are defined in `model_routine_mech_intrct.cpp`.

1. *computeForceSpAgent*: Compute forces between pairs of discrete agents (*see* Code 3). This model routine is called once per cell pair that is within the maximum direct mechanical interaction distance, at every *baseline time step*. Force on two interacting cells is set based on the overlap between the two cells to remove the overlap by pushing the two cells apart.

```
   void ModelRoutine::updateSpAgentBirthDeath(c
onst VIdx& vIdx, const SpAgent&
   spAgent, const AgentMechIntrctData& mechIn-
trctData, const Vector<NbrBox<
```

```
   REAL> >& v_gridPhiNbrBox/*[elemIdx]
*/, const Vector<NbrBox<REAL> >&
   v_gridModelRealNbrBox/*[elemIdx]
*/, const Vector<NbrBox<S32> >&
   v_gridModelIntNbrBox/*[elemIdx]
*/, BOOL& divide, BOOL& disappear) {
   /* MODEL START */
   divide=false;
   disappear=false;
   if((spAgent.state.getType()==AGENT_TYPE_R_
CELL)||(spAgent.state.
   getType()==AGENT_TYPE_G_CELL))
{/* R or G straincells*/
   if (spAgent.state.getRadius
()>= MAX_CELL_RADIUS){
   divide = true;
  }
 }
   else {/* dead cell */
   CHECK(spAgent.state.getType()== AGENT_
TYPE_D_CELL);
    }
   /* MODEL END */
   return;
 }
```

Code 2: Model routine to set whether a cell will divide, disappear, or neither divide nor disappear.

3.4 State Changes in the Extracellular Space

Model routines related to simulating state changes in the extracellular space are defined in `model_routine_grid.cpp`.

1. *initIfGridVar* and *initPDEBufferPhi*: Initialize grid state variables for the interface grid (*initIfGridVar*) and the PDE buffer grid (*initPDEBufferPhi*). *initIfGridVar* is called once per grid box in the interface grid, and *initPDEBufferPhi* is called once per grid box in the PDE buffer grid.
2. *updateIfGridVar*: Update interface grid state variables based on model specific rules (*see* Subheading 1.3). This function is called once per grid box in the interface grid.

```
   void ModelRoutine:: computeForceSpAgent(const
VIdx& vIdx0, const SpAgent&
   spAgent0, const VIdx& vIdx1, const SpAgent&
spAgent1, const VReal& dir/*
   unit direction vector from spAgent1 to spA-
gent0 */, const REAL& dist,
   VReal& force/* force on spAgent0 due to inter-
action with spAgent1 (force
   on spAgent1 due to interaction with spAgent0
has the same magnitude but
   the opposite direction), if force has the
same direction with dir, two
   cells push each other, if has the opposite
direction, two cells pull each
   other. */) {
   /* MODEL START */
   REAL R = spAgent0.state.getRadius () + spA-
gent1.state.getRadius ();
   REAL mag;/* + for repulsive force, - for
adhesive force */
   if (dist < = R) {/* shoving to remove the
overlap */
     mag = 0.5 * (R - dist);
    }
   else {/* adhesion */
     mag = 0.0;/* no adhesion */
    }
   for(S32 dim = 0; dim < DIMENSION; dim++) {
     force[dim] = mag * dir[dim];
    }
   /* MODEL END */
   return;
  }
```

Code 3: Model routine to compute force between two interacting discrete agents.

3. *updateIfGridKappa* and *updatePDEBufferKappa*: Set PDE parameter κ for the interface grid and the PDE buffer grid for each grid box. κ represents the cell volume exclusion in diffusion. As we are not considering cell volume exclusion, κ is set to 1.0 (0 % volume exclusion).
4. *updateIfGridAlpha* and *updatePDEBufferAlpha*: Set PDE parameter α. α sets the decay rate. We ignore lysine and adenine decay and set α to 0.0.
5. *updateIfGridBetaInIfRegion*, *updateIfGridBetaPDEBufferBdry*, *updateIfGridBetaDomainBdry*, *updatePDEBufferBetaInPDEBufferRegion*, and *updatePDEBufferBetaDomainBdry*: Set PDE parameter β. β sets the diffusion coefficient. β is set between two grid boxes sharing a face (*updateIfGridBetaInIfRegion* and *updatePDEBufferBetaInPDEBufferRegion*). Different model routines are called at the boundary between the interface grid and the PDE buffer grid (*updateIfGridBetaPDEBufferBdry*) and the simulation domain boundary (*updateIfGridBetaDomainBdry* and *updatePDEBufferBetaDomainBdry*). Code 4 sets the diffusion coefficient between two adjacent grid boxes in the interface

grid based on the yeast patterning model specifics by filling the function body of the predefined *Biocellion* model routine (*updateIfGridBetaInIfRegion*).

6. *updateIfGridRHSLinear* and *updatePDEBufferRHSLinear*: Set the PDE reaction term. *updateIfGridRHSLinear* sets the reaction term based on the lysine and adenine production and consumption rates for a grid box in the interface grid. No cells reside in the PDE buffer region and *updatePDEBufferRHSLinear* sets the reaction term to 0.0.
7. *updateIfGridAMRTags*: In solving PDEs, *Biocellion* users can apply different grid spacings for different regions; the finest grid spacing is the interface grid spacing. Users tag each box in the interface region with a desired AMR level. Boxes in the PDE buffer region are assumed to be tagged with the coarsest level. *Biocellion* (using CHOMBO [7]) generates an AMR hierarchy based on this information. We tag the boxes containing cells and the top 40 μm (in the *z* direction) in the agarose cylinder with the finest level. We tag the remaining boxes with the coarsest level.

3.5 Simulation Output

Model routines controlling simulation outputs are defined in `model_routine_output.cpp`.

1. *updateSpAgentOutput*: Color discrete agents. We color each discrete agent based on the cell type. We do not need to update extra output variables as we are mapping a discrete agent to a sphere. *See* Code 5 for our implementation for the *Biocellion* framework.

```
   void ModelRoutine::updateIfGridBetaInIfRegion
(const S32 elemIdx, const S32
   dim, const VIdx& vIdx0, const VIdx& vIdx1,
const UBAgentData&
   ubAgentData0, const UBAgentData& ubAgent-
Data1, const Vector<REAL>&
   v_gridPhi0, const Vector<REAL>& v_gridPhi1,
const Vector<REAL>&
   v_gridModelReal0, const Vector<REAL>& v_
gridModelReal1, const Vector<S32
   >& v_gridModelInt0, const Vector<S32>& v_
gridModelInt1, REAL& gridBeta)
    {
   /* MODEL START */
   REAL z0 = ((REAL)vIdx0[2] + 0.5) *
IF_GRID_SPACING;
   REAL z1 = ((REAL)vIdx1[2] + 0.5) *
IF_GRID_SPACING;
   REAL gridBeta0;
   REAL gridBeta1;
   if(z0 < AGAR_HEIGHT) {
   gridBeta0 = A_DIFFUSION_COEFF_AGAR[elemIdx];
   }
   else {
   REAL scale = (REAL)ubAgentData0.v_spAgent.
size ()/(REAL)
     UB_FULL_CELL_CNT;
   if(scale > 1.0) {
     scale = 1.0;
   }
   gridBeta0=A_DIFFUSION_COEFF_COLONY[elemIdx]
* scale;
   }
   if(z1 < AGAR_HEIGHT) {
   gridBeta1 = A_DIFFUSION_COEFF_AGAR[elemIdx];
  }
   else {
   REAL scale = (REAL) ubAgentData1.v_spAgent.
size ()/(REAL)
      UB_FULL_CELL_CNT;
   if(scale > 1.0) {
   scale = 1.0;
   }
   gridBeta1=A_DIFFUSION_COEFF_COLONY[elemIdx]
* scale;
   }
   if((gridBeta0 > 0.0) && (gridBeta1 > 0.0)) {
   gridBeta = 1.0/((1.0/gridBeta0 + 1.0/grid-
Beta1) * 0.5);/*
       harmonic mean */
   }
   else {
     gridBeta = 0.0;
   }
   /* MODEL END */
   return;
   }
```

Code 4: Model routine to set diffusion coefficient between two adjacent grid boxes in the interface grid.

```
   void ModelRoutine::updateSpAgentOutput(const
VIdx& vIdx, const SpAgent&
   spAgent, REAL& color, Vector<REAL>&
v_extra) {
   /* MODEL START */
   color = spAgent.state.getType ();
   CHECK (v_extra.size () == 0);
```

```
  /* MODEL END */
  return;
}
```

Code 5: Model routine to set the discrete agent color variable (for visualization).

3.6 Setup for a Simulation Instance

Biocellion asks users to provide specifics of a simulation instance (e.g., simulation domain size, output directory) in an xml file. See `$BIOCELLION_ROOT/framework/main/yeast-patterning-5um.xml`, `$BIOCELLION_ROOT/framework/main/yeast-patterning-20um.xml`, or `$BIOCELLION_ROOT/framework/main/yeast-patterning-40um.xml` for examples (for 5, 20, or 40 μm grid spacing, respectively).

We set the required parameters first.

1. Set the number of base line steps to execute. We set this number to 60,000 (500 h, <*time_step num_baseline* = *"60000"*/>).
2. Set the simulation domain size (<*domain x* = *"128" y* = *"128" z* = *"4864"*/> in case we adopt 5 μm interface grid spacing). As we are using three AMR levels with the refinement ratio of 4 (with 5 μm interface grid spacing, refinement ratio is also set in this xml file), simulation domain size should be a multiple of 64. We set the domain size in the *x* and *y* directions slightly smaller than the size in [5], while setting the size in the *z* direction slightly larger than the size used in [5].
3. Set the simulation initialization method and the partition size. We set the initialization method to initialize within the code, and set the partition size to 64 (<*init_data partition_size* = *"64" src* = *"code"*/>). Alternatively, users can start from checkpoint data.
4. Set the output directory path, interval, file formats, and the number of extra output variables for each discrete agent (<*output path* = *"output_directory_path" interval* = *"120" particle* = *"pvtu" num_extra* = *"0" grid* = *"vtm"*/>).

We also set several optional parameters relevant to executing the yeast patterning model on multicore PCs.

1. Set standard output verbosity (from 0 to 5) to 1 (<*stdout verbosity* = *"1"*/>).
2. Set the number of threads to 12 for a multicore PC with 12 hardware threads (<*system num_node_groups* = *"1" num_nodes_per_group* = *"1" num_sockets_per_node* = *"1" max_load_imbalance* = *"1.2' num_thre ads* = *"12"*>). *num_node_groups*, *num_nodes_per_group*, *num_sockets_per_node* and *max_load_imbalance* are irrelevant for multicore PCs and are ignored.

3. Set the summary report (*Biocellion* allows users to print the summary of the interface grid variables, see the *Biocellion* user manual [6] for additional details), AMR regridding, and checkpoint intervals (<*interval summary* = *"10" load_balance* = *"120" regridding* = *"120" checkpoint* = *"600"*/>). *load_balance* is irrelevant to multicore PCs and is ignored.
4. Set the refinement ratio in applying AMR and other parameters controlling AMR hierarchy generation (<*amr refine_ratio* = *"4"fill_ratio* = *"0.5"*/>). See the *Biocellion* user manual [6] for additional details.
5. Set the optional parameters affecting the accuracy of the multigrid method used in solving PDEs (<*mg_parabolic_solve mg_num_pre* = *"3" mg_num_post* = *"3" mg_num_bottom* = *"3" mg_v_or_w* = *"v" mg_max_ite rations* = *"50" mg_epsilon* = *"-12"mg_hang* = *"-8" mg_norm_threshold* = *"-20"*/>). See the *Biocellion* user manual [6] for additional details.

4 Notes

1. *Biocellion* users can set the framework to perform checks on model routine outputs or input arguments of *Biocellion* utility functions called inside model routines by enabling *CHECK_FLAG* = *-DENABLE_-CHECK* = *1* in `$BIOCELLION_ROOT/Makefile.model`. This often allows users to easily identify model program bugs such as using a random number generator without initialization or accessing a C++ STL vector array variable outside the array length. We recommend *Biocellion* users enable this option to verify their model routines prior to running full simulations. Once the model routines are verified, users should disable this check to expedite simulation.
2. Lysine released by a dying *G* strain cell is spread to seven grid boxes. The sum of lysine uptake and secretion rates for a grid box can be updated by seven different model routine calls (a model routine to update grid state variables is invoked once for every grid box in the interface grid in a single round). *Biocellion* asks users to set the synchronization method to properly update grid variables when a single variable is updated by multiple model routine calls. We set the synchronization method to `SYNC_METHOD_DELTA` to set the value by summing the differences from the initial value when updated by multiple model routine calls—e.g., if three different model routines set the value of variable rhs_{lysine} to 3, 5, and 9, respectively, then rhs_{lysine} is set to 3+5+9 (assuming that the initial value is 0). We need to reset the sum of lysine uptake and secretion rates (rhs_{lysine}) to 0.0 to ensure that this scheme properly works. We configure *Biocellion* to invoke model routines to edit grid variables at the end of a *state-and-grid time step* to reset the variable to 0.0.

Acknowledgements

Support for this research was provided by the Extreme Scale Computing Initiative and the Fundamental and Computational Sciences Directorate, as part of the Laboratory Directed Research and Development Program at Pacific Northwest National Laboratory (PNNL). Portions of this work were conducted using PNNL Institutional Computing at PNNL. PNNL is operated by Battelle for DOE under contract DE-ACO5-76RLO 1830. B.M. is a Gordon and Betty Moore Foundation fellow of the Life Sciences Research Foundation.

References

1. Byrne H, Drasdo D (2009) Individual-based and continuum models of growing cell populations: a comparison. J Math Biol 58(4–5):657–687
2. Colella P, Graves DT, Johnson JN, Johansen HS, Keen ND, Ligocki TJ, Martin DF, McCorquodale PW, Modiano D, Schwartz PO, Sternberg TD, Van Straalen B (2012) Chombo software package for AMR applications design document. Lawrence Berkeley National Laboratory, Berkeley, CA
3. Ferrer J, Prats C, López D (2008) Individual-based modelling: an essential tool for microbiology. J Biol Phys 34(1–2):19–37
4. Galle J, Loeffler M, Drasdo D (2005) Modeling the effect of deregulated proliferation and apoptosis on the growth dynamics of epithelial cell populations in vitro. Biophys J 88:62–75
5. Momeni B, Brileya KA, Fields MW, Shou W (2013) Strong inter-population cooperation leads to partner intermixing in microbial communities. Elife 2:e00230
6. Pacific Northwest National Laboratory (2013) Biocellion 1.0 User Manual, 1.0 edition, Accessed Jul 2013
7. Xavier JB, Picioreanu C, van Loosdrecht MCM (2005) A framework for multidimensional modelling of activity and structure of multispecies biofilms. Environ Microbiol 7(8): 1085–1103