A parallel implementation of an off-lattice individual-based model of multicellular populations

Daniel G. Harvey[a], Alexander G. Fletcher[b], James M. Osborne[a,c], Joe Pitt-Francis[a,*]

[a] *Computational Biology Group, Department of Computer Science, University of Oxford, Parks Road, Oxford, OX1 3QD, UK*

[b] *Wolfson Centre for Mathematical Biology, Mathematical Institute, University of Oxford, Andrew Wiles Building, Radcliffe Observatory Quarter, Woodstock Road, Oxford, OX2 6GG, UK*

[c] *Computational Science, Microsoft Research, 21 Station Road, Cambridge, CB1 2FB, UK*

Abstract

As computational models of multicellular populations include ever more detailed descriptions of biophysical and biochemical processes, the computational cost of simulating such models limits their ability to generate novel scientific hypotheses and testable predictions. While developments in microchip technology continue to increase the power of individual processors, parallel computing offers an immediate increase in available processing power. To make full use of parallel computing technology, it is necessary to develop specialised algorithms. To this end, we present a parallel algorithm for a class of off-lattice individual-based models of multicellular populations. The algorithm divides the spatial domain between computing processes and comprises communication routines that ensure the model is correctly simulated on multiple processors. The parallel algorithm is shown to accurately reproduce the results of a deterministic simulation performed using a pre-existing serial implementation. We test the scaling of computation time, memory use and load balancing as more processes are used to simulate a cell population of

*Corresponding author.

Email addresses: danielharvey458@gmail.com (Daniel G. Harvey), alexander.fletcher@maths.ox.ac.uk (Alexander G. Fletcher), james.osborne@cs.ox.ac.uk (James M. Osborne), joe.pitt-francis@cs.ox.ac.uk (Joe Pitt-Francis)

fixed size. We find approximate linear scaling of both speed-up and memory consumption on up to 32 processor cores. Dynamic load balancing is shown to provide speed-up for non-regular spatial distributions of cells in the case of a growing population.

Keywords: computational biology; individual-based models; off-lattice; parallelisation; scaling analysis

1. Introduction

Individual-based simulation enables the modelling of complex systems, such as multicellular populations in biology, at increasingly detailed resolution [1]. An advantage of this approach is that the collective behaviour emerging from the interactions of heterogeneous individuals can be studied in a way that is not possible using classical methods. Furthermore, it is often possible to parameterise individual-based models using quantities that may be more readily measured experimentally, leading to an increasingly symbiotic relationship between modelling and the experimental sciences.

There are, however, a number of difficulties with the individual-based approach. One practical limitation is the computational cost of simulating such models. Simulations typically consist of a large number of interacting individuals, each of which has an internal state that evolves in response to interactions with other individuals according to pre-determined rules. The need to store the state of each of the n individuals in a simulation leads to a memory requirement that grows as $\mathcal{O}(n)$. Worse, for pairwise interactions between individuals, the processing load grows as $\mathcal{O}(n^2)$. This is a problem that is keenly felt in individual-based modelling of multicellular populations, where each cell is represented as a discrete entity. For large systems, computational cost can make simulations intractable, either as a result of excessive memory consumption or excessive processing time.

A common approach in the face of such computational limitations is to employ a continuum model. While it is possible to derive continuum models in certain circumstances from their individual-based counterparts under certain limits using coarse-graining techniques [2, 3, 4], or to combine continuum and discrete models in a hybrid system [5, 6], such approaches are largely restricted to simplifications of the original model and many of the benefits of the individual-based approach are consequently lost. By increasing the computational resources available for simulation, it is possible to signif-

icantly increase the scale of model that may be practically simulated using an individual-based approach without placing restrictions on the modelling methodology.

Parallel computing technology enables multiple single computational units to execute a program, such as a simulation of an individual-based model. This affords a significant increase in the available memory and computing power for a single simulation, potentially increasing the scale of simulation that is possible. The fact that such an approach has seen limited use in the field of individual-based modelling of cell populations in biology may be due to the difficulty in constructing computational algorithms that may be executed on parallel computing technology. The typical development cycle of software for such simulations is to begin with a simple serial algorithm, which is iteratively extended to more detailed modelling until the limits of computational tractability are hit. At this point it is often too costly to alter the software so that it may be executed in parallel, as there tend to be many explicit and implicit assumptions of serial execution within the program.

1.1. Individual cell-based modelling approaches

A variety of different discrete modelling approaches have been developed to describe how individual cells interact within a population. These range in complexity from simple cellular automata, where each cell is represented by a single site on a lattice, with cell state updated by pre-defined rules, to physical models where individual cells are partitioned into mesoscopic elements to better capture cell mechanics and deformation.

By imposing that the physical representation of cells is restricted to a fixed geometric lattice and updating their location in terms of rules or the minimisation of a heuristic potential function, lattice-based models such as cellular automata [7, 8] and the cellular Potts model (CPM) [9, 10, 11] offer a simple description of the dynamics of cells within the population. This typically makes computation less expensive. However, parameters used to define such physical models are often difficult to relate to experimentally measurable quantities since they typically arise from heuristic rules. Furthermore, it is more difficult to define a time scale for such models so that simulation progression may be compared with temporal experimental data. In many cases these drawbacks are not critical to the biological hypotheses being investigated and lattice-based models provide an ideal framework. However, when the physical interaction of cells forms a key part of the model (for example,

in studying contact inhibition of cell growth [12]) an off-lattice approach is often favoured.

Common off-lattice approaches include cell-centre models, where each cell is represented by a single point in space with an associated volume (defined, for example, through a Voronoi tessellation), and vertex models, where each cell is represented by a polygon (or polyhedron), whose vertices are shared with neighbouring cells [13, 14]. Here we focus on developing an algorithm for cell-centre models, which have been used to study biological processes such as solid tumour development [12, 15], intestinal homoeostasis and carcinogenesis [16, 17, 18] and ductal carcinoma *in situ* [19].

1.2. Parallel approaches

Efforts to develop parallel algorithms for individual-based models of cell populations have largely been focused on the CPM. A parallel algorithm has been reported for the CPM that allows $\mathcal{O}(10^6)$ cells to be simulated on approximately 25 processors [20]. The authors decompose the lattice into spatial regions divided between the processors and employ a lock-based algorithm where neighbouring sites are locked (not updated) while a lattice site is being updated to ensure the model is correctly simulated in parallel. Although one might expect that imposing a wait on each processor using the locking algorithm during each update step would lead to large inefficiencies of the algorithm, efficiencies of between 80% and 90% are reported for simulations of $\mathcal{O}(10^6)$ cells. Similar approaches, based on decomposing space between processors, have been used to parallelise the algorithms of related lattice-based models [21, 22].

Another notable example, from the wider field of agent-based modelling of multicellular populations, is the parallel functionality provided by the Flexible Large-scale Agent-based Modelling Environment (FLAME) [23]. Using the Message Passing Interface (MPI), FLAME has demonstrated 80% parallel efficiency on over 400 processes for a benchmark simulation, although the authors note that decomposing agents based on their state variables (for example, a space decomposition) gives rise to load balancing problems, where different processes may be responsible for a disproportionate amount of computation [24].

Some recent developments in parallel cell population simulations have focused on the use of General Purpose Graphics Processing Units (GPGPUs), which require modest capital outlay and achieve significant performance improvements over serial CPU algorithms. For example, a GPGPU approach

has been shown to substantially improve the performance of the subcellular element model [25], a class of off-lattice model that includes a fine-grained description of cell mechanics [26]. By moving computationally costly tasks, such as neighbour identification, to the GPGPU the authors were able to achieve an 18-fold increase in performance. Nevertheless, they were only able to report on simulations where the total number of cells grew to approximately 5,000, and such simulations took almost 10 hours to compute. In other work a simulation of a cell population has been implemented in FLAME using a GPGPU combined with a spatial decomposition approach, obtaining speed-ups of 10,000 [27]. While such large speed-ups are available from GPGPU technology, they are limited in their memory capacity, and for detailed models of individual agents, may still pose a high communication overhead when transferring agents from the CPU to the GPGPU.

The greatest progress in the use of high-performance computing technology in cell-based modelling has come in the lattice-based paradigm. It has been suggested that simulating individual-based models with a high degree of connectivity between the individuals on distributed HPCs is ineffective due to communication latency and bandwidth restrictions, and that shared-memory approaches such as GPGPUs are of greater benefit [22]. However, GPGPU technology, while providing valuable speed-up for relatively small simulations, lacks the memory and processing power to tackle new scales of problem. On the other hand it has been shown that distributed HPC methods can be a viable option for large-scale simulation of cell population models [20].

To address the lack of suitable parallel methodologies for off-lattice, individual-based models of cell populations, in this paper we describe a novel parallel algorithm for a class of such models. Our algorithm combines techniques for domain decomposition from computational physics with a novel load balancing algorithm that improves efficiency of simulations where the spatial domain is not known ahead of runtime. In contrast to GPGPU or shared memory approaches, which are tied to a specific computational system design, we choose to use the architecture-independent Message Passing Interface (MPI) to allow for an implementation that is portable between multi-threaded and multi-core shared-memory and grid-based distributed memory parallel computing systems. By implementing inter-process communication in a model-agnostic manner, our implementation is able to support parallel simulations of unbounded variety by modification of the individual-cell model, and of the cell-cell interaction rules. Our parallel algorithm only

places the restriction of spatial locality on cell-cell interaction laws.

The remainder of this work is structured as follows. In Section 2 we develop a parallel algorithm and implementation for a class of individual cell-based model that is available as part of the open-source Chaste project [28]. The implementation of the algorithm is sufficiently flexible and extensible that it may be applied to a wide range of problems in biological modelling, by changing the rules governing the behaviour and interactions between individual cells in the model. In Section 3 we present results concerning the validation of our algorithm and its performance. In Section 4 we conclude with a summary of our findings and suggestions for further work.

2. Methods

2.1. Individual-based model

The model for which we have developed our algorithm treats each cell as an individual agent that contains data defining its present state. In its simplest form, these data are each cell's location (which is not restricted to a lattice, and may be modelled in either 1, 2 or 3 spatial dimensions). However, other state information relating to the cell may also be stored, such as the cell's progress through its cell cycle, or its intracellular levels of specific metabolites or signalling factors. Physical interactions between cells are assumed to be governed by a pairwise force law that is short-ranged in nature (that is, only acting between cells that are separated by less than some fixed distance), while cells may also interact with neighbouring cells through other mechanisms, such as lateral signalling, where a cell's behaviour may be influenced by the concentration of a signalling factor in nearby cells.

The dynamics of the cell population are governed by a system of Langevin equations of motions for the cells, based on a force-balance approach [29]. Assuming that a cell i located at $\mathbf{r}_i$ experiences a force $\mathbf{F}_{ij}$ from each neighbouring cell j, the motion of cell i is governed by

$$m_i \frac{d^2\mathbf{r}_i}{dt^2} = -\eta_i \frac{d\mathbf{r}_i}{dt} + \sum_{j=0}^{N_i-1} \mathbf{F}_{ij}, \tag{1}$$

where η_i is a viscous drag constant describing the bulk viscosity of the surrounding media (neighbouring cells and extra-cellular matter), m_i is the mass of cell i, and the sum is taken over all N_i neighbouring cells, defined as those

cells for which $\|\mathbf{r}_i - \mathbf{r}_k\|_2 < r_c$ for some fixed constant r_c, the maximum interaction radius. This is often termed an 'overlapping spheres approach' to modelling cell neighbours. We assume that the Reynolds number for the motion of each cell is small, so that viscous forces may be assumed to dominate inertial forces [30]. We thus approximate (1) by the first-order equation

$$\frac{d\mathbf{r}_i}{dt} = \frac{1}{\eta_i} \sum_{j=0}^{N_i-1} \mathbf{F}_{ij}. \tag{2}$$

From the perspective of our parallel algorithm, models at the subcellular scale are restricted in that they must only depend on the data from neighbouring cells. The algorithm may be applied to simple rule-based subcellular models, or to more complex systems of differential equations describing the state of subcellular gene networks. The subcellular model may interact with the physical model either through the modification of the cell-specific parameters defining the interaction force between cells or by introducing a new cell into the population as the result of a cell division, or by removing cells from the population through cell death.

2.2. Model simulation

We evolve the simulation in discrete time steps of constant size Δt. Here we describe the approach to updating the location of each cell in space, since this is key to the development of the parallel algorithm for simulation due to its dependence on the data from neighbouring cells. Updating the state of subcellular models (such as the cell-cycle or internal gene networks) that depend on data from neighbouring cells is analogous to the physical model.

Each cell i is assigned an initial location $\mathbf{r}_i(t_0)$. Using a forward Euler scheme to approximate the solution of equation (2), we update the location of each cell i at time $t_{n+1} = t_n + \Delta t$ from $\mathbf{r}_i(t_n)$ to

$$\mathbf{r}_i(t_{n+1}) = \mathbf{r}_i(t_n) + \frac{\Delta t}{\eta_i} \sum_{j=0}^{N_i-1} \mathbf{F}_{ij}(t_n). \tag{3}$$

The solution of this equation depends on computing the force components $\mathbf{F}_{ij}$ for each neighbouring cell j. These forces typically depend on the states of cells i and j (for example, their locations, size, or subcellular states). It is this dependence of the motion of each cell on its neighbours that motivates our division of computational tasks in a parallel environment.

2.3. Parallel decomposition

An optimal decomposition of a serial algorithm into parallel tasks will split the algorithm so that each component may be carried out independently of the others. Such a decomposition leads to an optimal increase in performance of the computation of the algorithm. Dependence of separate parallel tasks can create both a communication overhead between the separate processes executing each task (which may be located on separate processors), as well as inefficient execution when one task must wait for another to complete. Due to the dependence of the model of each cell on its neighbours, it is not in general possible to provide an independent decomposition for our cell population model and simulation. We therefore aim to minimise the dependence between separate tasks.

The most natural unit of decomposition within the model is the single cell: the evolution of the subcellular model over each time step provides an independent task that may be carried out by separate processes. As outlined in Section 2.1, the model of each cell depends only on data from other cells within close spatial proximity, and this motivates us to divide cells between separate processes based on their spatial location. That is, separate processes are each assigned a subset of physical space and are responsible for simulating the cells that lie in that region of space. Alternative approaches have been developed in the field of particle simulations, such as molecular dynamics. For example, an arbitrary decomposition between processes can be a successful approach where data dependence is low, and the cost of communication between the processes is low, with further optimisations possible by using the symmetry of force between two particles or cells. Although spatial decomposition algorithms have also been suggested for molecular dynamics, the relatively small amount of data associated with a single particle makes the approach less favourable than arbitrary decompositions [31]. As a result of the level of detail that can potentially be included within a single cell in our model, a spatial decomposition is likely to offer a more efficient parallel algorithm as it will only require data communication between a small subset of the processes being used.

Figure 1 shows the spatial decomposition scheme chosen for our algorithm in the case of two spatial dimensions. We split the spatial domain into axis-aligned rectangles with a minimum linear dimension of r_c along each axis. Consequently there are two regions in each spatial domain where a cell may depend on data from a neighbouring process. For example, in Figure 1, Cell 1 lies in the shaded region C, which lies within a distance r_c of the

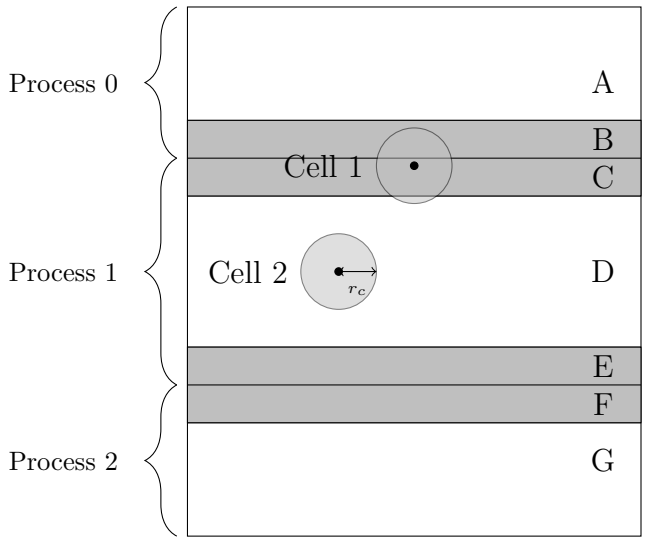


Figure 1: Decomposition of a two-dimensional physical domain between different processes in parallel. Each process owns one rectangular region of space, and is responsible for updating cells in that region. Data for cells less than a distance r_c (the maximum interaction radius) from the boundary must be exchanged between neighbouring cells at each time step.

boundary between Process 1 and Process 2. Before the calculation of each time iteration, data must be exchanged from Process 1 and Process 2 on all cells in region C, and likewise for all other neighbouring processes. After updating the state of each cell and integrating the equations of motion over the time step to update the location of each cell it is necessary to re-assign cells between processes so that the data and ownership of each simulated cell is stored in the memory of the process owning the region of space containing its centre.

In order to obtain optimal computation speed and scale it is necessary to make full use of each computing resource when a model is being simulated in parallel: this requires that an approximately equal proportion of computational tasks is assigned to each resource at any point during the simulation. In terms of simulation execution time, the observed wall-clock duration of a simulation is that of the last resource to complete its tasks. Consequently, an overloaded process which will take proportionally longer to complete its tasks will limit execution speed as other processes sit idle waiting for it to

complete. In our algorithm, computational tasks are assigned based on the spatial distribution of cells, and it is not therefore guaranteed, or indeed likely, that an even distribution of computational load will be achieved.

To address this, we implement a simple dynamic load-balancing algorithm with the implementation of our parallel algorithm, that periodically re-distributes the model domain between processes to maintain an approximately uniform distribution of cells between processes. Figure 2 depicts how a region containing a larger number of cells compared to its immediate neighbours, and therefore under higher computational load, has its domain shrunk with its neighbours being assigned a larger spatial region, and therefore more cells.

To decide whether, and in which direction, to move each boundary, we consider the number of cells in each strip of height r_c within each domain. One of these strips is exchanged with a neighbouring process (by moving the cell data onto that resource) if doing so reduces the imbalance between the two resources. In particular, assuming each region of space assigned to a process labelled p is divided into S_p sub-regions of size r_c, with each sub-region i containing w_i^p cells, then we quantify the load imbalance σ_{pq} between two processes p and q as the squared difference in cell numbers contained on each process:

$$\sigma_{pq} = \left(\sum_{i=0}^{S_p-1} w_i^p - \sum_{i=0}^{S_q-1} w_i^q \right)^2 . \tag{4}$$

For each pair of processes we calculate the change in σ_{pq} if a single neighbouring strip of size r_c is exchanged in each direction (i.e. p sends one strip to q, q sends one strip to p or no change). The exchange that results in the greatest decrease in σ_{pq} is chosen.

This algorithm is carried out every K integration time steps throughout the algorithm, where K is a user-defined simulation parameter that may be tuned for optimal performance. We investigate the appropriate value of K and the sensitivity of the simulation performance to its value in Section 3.3.

2.4. Implementation

The Chaste library is implemented in `C++`, with parallel communication between processes implemented using the Message Passing Interface (MPI) [32]. `C++` provides a number of benefits over other suitable languages such as Java or Python. In particular, code optimisations implemented by

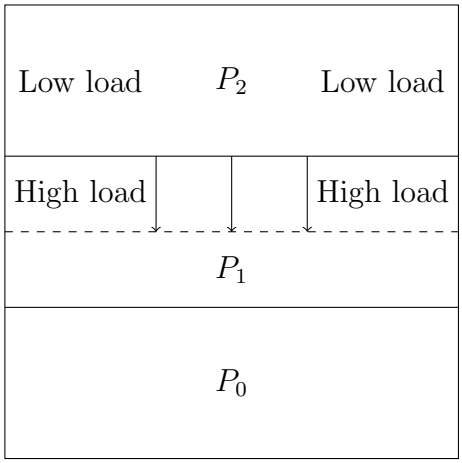


Figure 2: Load balance is achieved by shifting the boundaries of process domains to minimise local pairwise imbalances. In this case the boundary is shifted so that P_2, a lower loaded process, takes some of the load from P_1.

most modern C++ compilers can lead to extremely efficient runtime performance, while explicit memory management allows for effective use of the processes available.

MPI provides a highly portable parallel communication model that may be used across both shared- and distributed-memory parallel computers and is therefore well suited to the general purpose nature of the Chaste library. In contrast other approaches are typically restricted to one type of parallel architecture. For example, multithreading models are restricted to shared-memory machines (where all processes share a single memory block), while GPGPUs require both an on-board graphics card as well as typically non-portable hardware-specific code.

To support the extensible implementation of the model of each single cell provided by Chaste (where any user-defined model may be created) within the parallel framework, we use the Boost Serialization library [33] to turn a C++ object that represents a single cell into a binary string that may be sent between processes using MPI. This implementation means that future cell models not currently implemented within the Chaste library will be supported by our parallel implementation without additional modifications to the communication algorithms. The parallelisation of the individual-based model forms part of Chaste open source release 3.2 and any additional code

or data required to reproduce the results in this paper is available from our website[1].

3. Results

3.1. Validation

To confirm that our parallel algorithm, and its implementation in Chaste, correctly simulates the cell dynamics of the model, we numerically compare the results of a simulation of simple cell-cell interaction as well as a larger, deterministic simulation with the result of the same simulation execution in serial.

We first consider the case of three cells pushing apart due to their cell-cell interaction force. In parallel, one cell is located on a separate computational unit from the other two. The cells are initially in close proximity, with cells located at $(0, 0)$, $(0, 0.4)$ and $(0, 0.6)$, and at each numerical time step we record the location of each cell to full machine precision. We repeat this process for a 2D simulation of 256 cells arranged initially in close proximity, where the population is allowed to relax into physical equilibrium for 100 hours of simulation time. In each case we calculate

$$\sigma_n = \frac{1}{N_i} \sum_{j=0}^{N_i-1} \|\mathbf{x}_i^s(t_n) - \mathbf{x}_i^p(t_n)\|, \tag{5}$$

the mean difference at each time step between a cell's location in the serial simulation ($\mathbf{x}_i^s$) and the equivalent cell's location in the parallel simulation ($\mathbf{x}_i^p$). The numerical results for the three-cell simulation were identical—that is $\sigma_n = 0$ for all n, when simulating on a 64-bit Linux desktop. For the larger simulation, in Figure 3(a) we plot the error after each hour of simulation time, relative to machine epsilon.

We see that the error grows from zero for the initial condition to $\mathcal{O}(10^{-15})$ after 100 hours of simulation time or 240,000 numerical time steps. We anticipate that this small discrepancy is due to changes in the order of summation in equation (3) when cells are located on different processes. When floating point numbers are summed in a different order, marginally different values can be obtained. In particular, the naïve method of summation employed by

[1] `https://chaste.cs.ox.ac.uk/trac/wiki/PaperTutorials/Harvey2015`

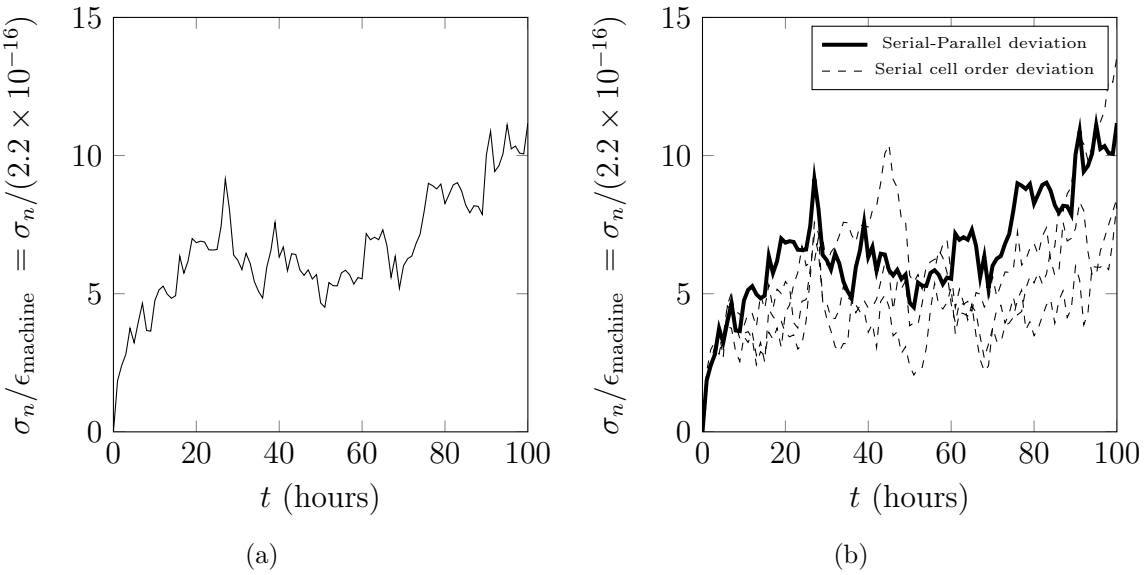


Figure 3: (a) Mean difference between cell locations from the serial and parallel simulations relative to machine precision for a deterministic cell population simulation. (b) The mean difference in cell locations plotted against the equivalent mean difference measure between two serial simulations where the internal order of cells in the data structures has been permuted.

Chaste for updating equation (3) is vulnerable to such inaccuracies as the partial sum grows larger than each individual summand. In order to show that our error is consistent with summation errors, in Figure 3(b) we plot a number of error traces for the same simulation in serial when we permute the order of summation by altering the order in which cells are stored internally within the data structures of the code. It can be seen that the error induced by our parallel algorithm is on the same order as that caused by these summation order errors. Overall, such a small difference represents a good agreement between the parallel and serial algorithms.

3.2. Performance

The objective of developing a parallel algorithm for cell-centre models is to increase the scope of simulations that are possible, as well as reducing the computational runtime of a given simulation. To test our implementation with respect to these objectives we carry out two benchmarking tests.

To assess the scalability of the parallel algorithm in terms of simulation

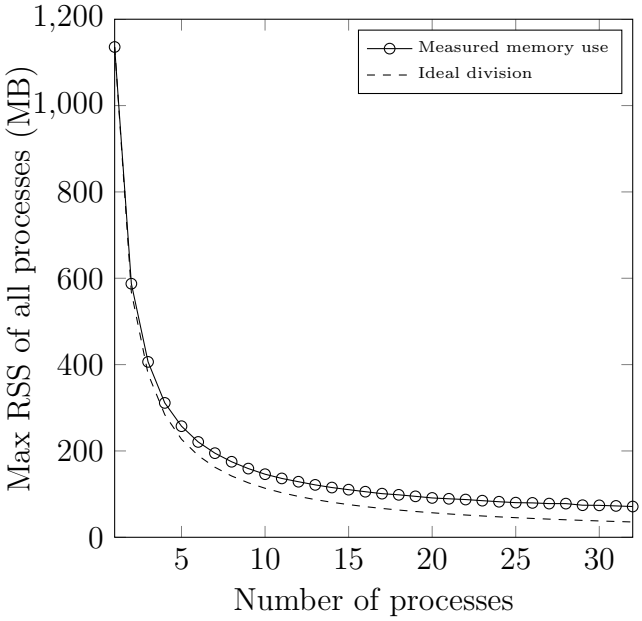


Figure 4: Memory use, measured using maximum resident set size over all processes for a simulation of 1,024,000 cells in a 2D monolayer.

size, we measure the memory usage of a fixed simulation as we introduce more processes up to the limit of the number of physical processors available. Whilst establishing the precise memory use of a program at runtime is not totally accurate or portable between different computer architectures, by measuring the resident set size (an approximation for total memory use) after initialising a simulation of 1,024,000 cells evenly distributed through the spatial domain, using calls to the C function `getrusage()`, we may obtain an estimate of the memory use of a simulation. Figure 4 shows the largest recorded resident set size across all processes for between 1 and 32 processes on a shared-memory computer, plotted alongside an idealised line showing the ratio of the resident set size recorded on one process to the number of processes used. The measured resident set size tracking closely to the ideal division line shows, for this simulation of evenly distributed cells, that our algorithm scales efficiently in terms of memory use, and the size of simulation that is computationally tractable will grow in proportion to the number of processes used.

Whilst memory consumption limits the number of cells that may be sim-

ulated, in practice the runtime for a large simulation must be tractable for it to be a realistic tool for investigating hypotheses. To assess the runtime performance of our parallel algorithm, we measure the wall-time (that is, the time experienced by a user) taken to execute a simulation of approximately 10^6 cells on a rectangular 2D monolayer as we increase the number of processes used. For this test we use between 1 and 32 cores of a SGI UV 100 shared-memory machine. To assess the scaling of the wall-time for the simulation we calculate the speed-up, defined as

$$S_p = \frac{T_p}{T_1}, \tag{6}$$

where T_1 is the total wall-time for the simulation to complete in serial (by one process), and T_p is the total wall-time for the simulation to complete on p processes. Optimal parallel scaling is achieved when $S_p \approx p$. In Figure 5 we show the results of our wall-time scaling experiment, plotting S_p for our simulation and, for reference, $S_p = p$. We see that, using up to 32 processor cores, we achieve a speed-up of 20.91, which reduces the wall-time for this simulation from approximately 2,000 seconds to around 100 seconds. Furthermore, we see the scaling is approximately linear over the range of processes available, with no plateau in the rate of speed-up achieved. However, the scaling of our simulation falls short of the optimal $S_p = p$ scaling, due to the necessity of communicating data of neighbouring cells between processes, which takes processes away from computations directly related to the simulation of the model.

3.3. Load balancing

To evaluate the performance of our dynamic load-balancing algorithm, we carry out a similar scaling experiment for a non-uniform, growing 2D monolayer of cells. The monolayer grows from approximately 25 cells to around 10^4 of the course of 100 hours of simulation, with cell growth and division occurring over a time scale of hours. The monolayer naturally assumes an approximately circular configuration, and grows in size over the course of the simulation, meaning that a static decomposition of the spatial domain would lead to a non-uniform distribution of cell number between processes.

In Figure 6(a) we plot the speed-up S_p for between 1 and 8 processes with and without the load-balancing procedure applied. When the load-balancing procedure is applied at a rate of once per hour of simulation time, the maximum speed-up attained increases from 1.1 to 2.9. Furthermore, it

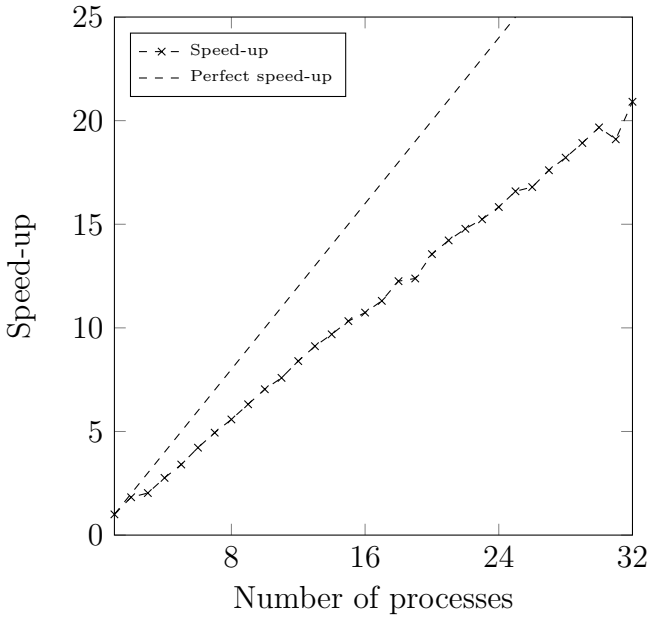


Figure 5: Speed-up of benchmark simulation on 1–32 cores of a shared memory system with one process assigned to each processor core. The dashed line shows idealised scaling, and the crossed line shows the scaling of our example simulation. The scaling is approximately linear over the range of processors available, obtaining a maximum speed-up of 20.91x on 32 processes.

can be seen in Figure 6(b), where we plot the final distribution of cell numbers at the end of the simulation, that the algorithm is successful in re-balancing the cells between 8 processes. The final distributions of cells from these two simulations are also shown graphically in Figure 7(a) (for no load-balancing) and Figure 7(b) (for dynamic load-balancing). In these figures each sphere is the final position for an individual cell in the growing two-dimensional sheet and the cells are coloured according to which of the 8 processes they lie on.

The choice of the frequency at which to apply the load-balancing algorithm depends largely on the rate of spatial redistribution of the cells, which drives movement from an evenly balanced state, to an unevenly balanced state. This can be seen in Figure 6(c), where we plot the total simulation time on 8 processes as a function of the rebalancing rate. For this simulation an optimal frequency is obtained at approximately one load-balance iteration per hour, which is at approximately the same rate as the growth of the cell population.

4. Discussion

In this work we have demonstrated an algorithm for the parallel simulation of a class of off-lattice model of multicellular populations. We have provided an implementation that demonstrates efficient scaling, both in terms of memory use and wall-clock execution time for a fixed simulation. Our work takes techniques for decomposing spatial simulations developed in other fields, such as molecular dynamics, and applies it to biological simulation where a challenge is presented by the level of detail associated with each cell in the model. We advance towards the goal of more detailed and larger simulation that can better capture the key biological processes of interest.

Our algorithm achieves close to perfect scaling of memory use for a spatially evenly distributed cell population. This, combined with our architecture-independent MPI implementation, creates the opportunity for simulating models at a scale that would otherwise be intractable. Furthermore, our algorithm not only demonstrates computation-time scaling efficiency of around 65%, which is comparable with parallel implementations of simpler, fixed-size lattice models [20] but also allows for more realistic cell dynamics that may be more easily related to experimental data as well as for dynamically growing populations of cells. While a static decomposition of space is appropriate for lattice-based models, cells in off-lattice models may rearrange with respect to their initial spatial configuration, leading to significant load imbalances

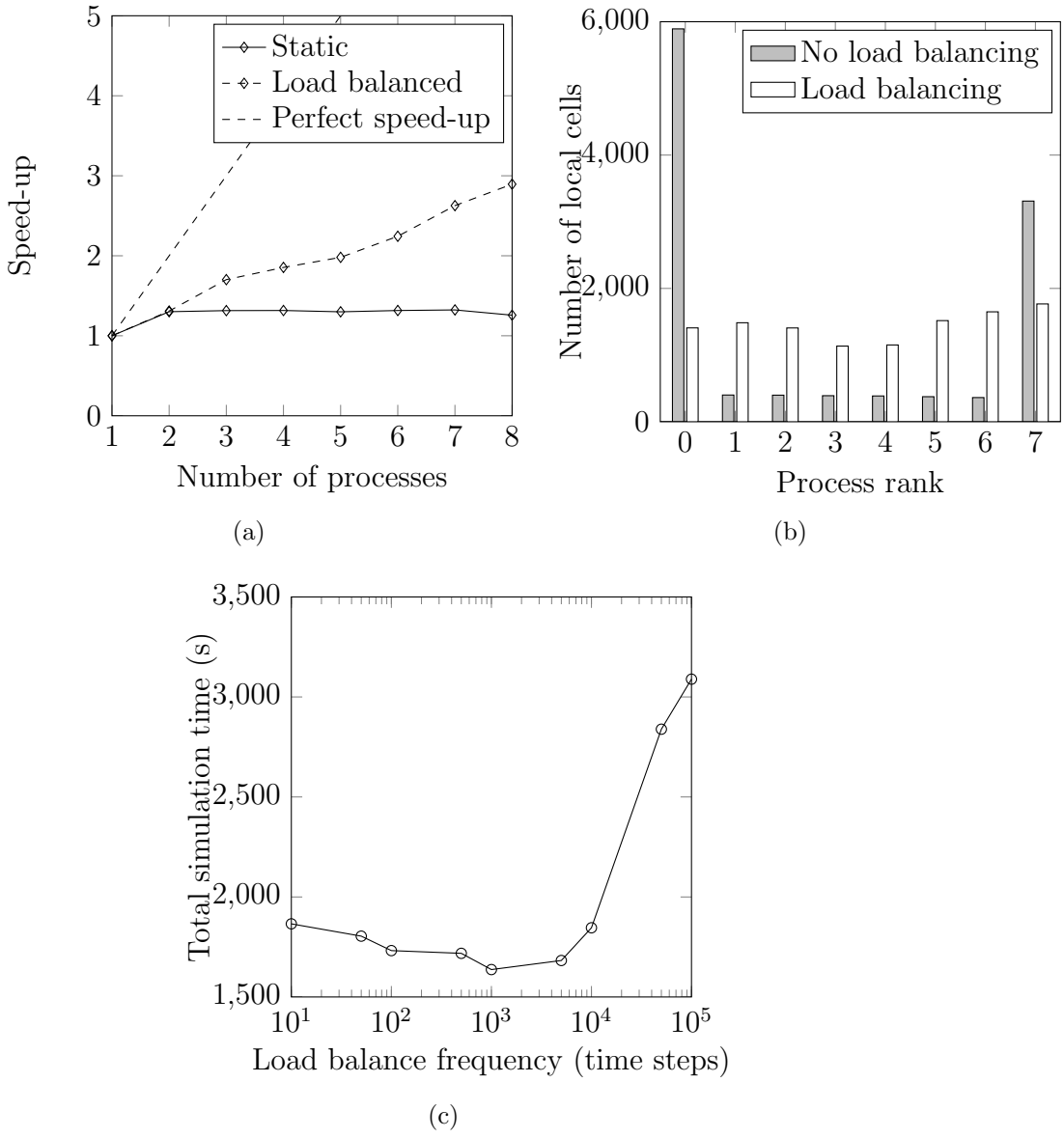
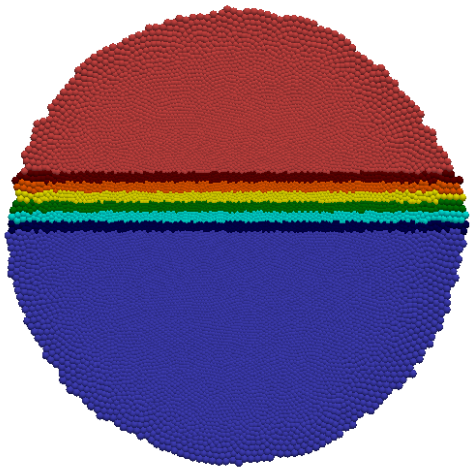
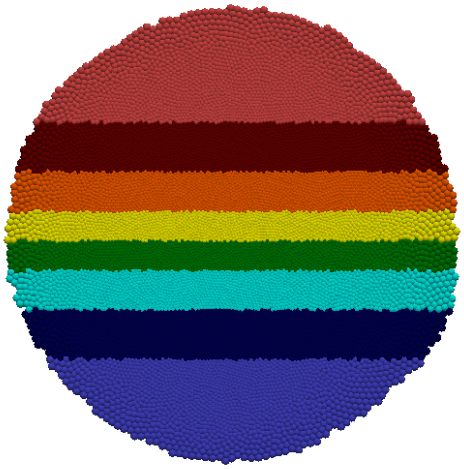


Figure 6: (a) Speed-up achieved using a dynamic load-balancing scheme in a simulation of a growing monolayer of cells. The dynamic load-balancing algorithm increases maximum speed-up from 1.3x to 2.9x over 8 processes. (b) The distribution of cell numbers between processes at the end of an 8-process simulation. The dynamic load-balancing algorithm leads to a more uniform distribution of cells between computational processes. (c) The sensitivity of the total computation time of the simulation to the rate of load-balancing. A greatest computation speed is achieved with a rebalance rate of approximately once per 10^3 s.



(a)



(b)

Figure 7: Snapshots of a simulation of a two-dimensional cell population, with cells coloured according to which process they lie on. (a) The distribution of cells without the application of the load-balancing algorithm. As the population grows, the extremal regions become unevenly loaded. (b) The distribution of cells in a simulation using the dynamic load-balancing algorithm.

and reduced parallel efficiency. We have mitigated this problem by developing a novel load-balancing algorithm that rebalances the spatial domain on approximately the time scale of growth within the population.

The efficiency of our algorithm will depend on a number of factors. Assuming a fixed underlying computational architecture, the quantity of data stored within a cell will affect performance. Cells with more detailed descriptions of proliferative behaviour and cell-cycle progression will be more expensive to serialise and communicate between compute resources. However, the level of complexity of the internal cell-cycle algorithm will not adversely affect parallel performance, since only cell state must be communicated between compute resources. Two distinct, but related, biological environmental factors further affect the parallel efficiency of our algorithm: cell density and cell motility. Cell density dictates the number of cells which fit into a strip of given width and therefore the number of positional updates which are exchanged between neighbouring processes. Thus increasing the cell density not only increases the number of cell-cell force calculations (an operation which is local to a process) but also increases the communication overhead (which may reduce the parallel efficiency). Cell motility also has potential to increase the communication overhead of our algorithm. If cells are more motile then they are more likely to move across process boundaries which is an operation that requires the cell data to be migrated from one process to another. At high motility rates the spatial load balancing step must be run more frequently due to changes in cell distribution.

A natural extension of our work is to further reduce the communication cost so that we can achieve wall-time speed-up. For specific model applications we could achieve this by specialising the data transmitted between processes, rather than sending all cell data. However, such a modification would need to be specialised for each application, and would not provide a general-purpose implementation.

A further possible direction of development is to consider the use of other parallel technologies. While our MPI implementation is highly portable between different shared and distributed-memory systems, GPGPUs could provide another approach to speed-up. As noted earlier, we must still copy cell data between the CPU and the GPGPU, but if such operations are faster than equivalent transfer of cell data through the network of a distributed system this approach may enable further speed-up.

Our load-balancing algorithm could be further developed in a number of ways. First, we could consider iterating the algorithm more than once at each

time step, with the aim of improving the load distribution. However, this would require a larger movement of cells between processes at each iteration, and the results shown in Figure 6(c) suggest that choosing a single iteration of the algorithm on the time scale of population growth and re-organisation is sufficient to maintain a broadly balanced load distribution. Second, we could also take account of communication in the measure of load. For example, by defining local load to be the sum of the number of local cells and cells in neighbouring regions that must be communicated, we may achieve an improved overall balance of computation time.

Acknowledgments

D.G.H. acknowledges funding by an EPSRC-funded Systems Biology Doctoral Training Centre Studentship (EP/G03706X/1). A.G.F. and J.M.O. are funded by the EPSRC (http://www.2020science.net) through grant EP/I017909/1. J.M.O. is also supported by Microsoft Research Cambridge.

References

[1] J. Galle, G. Aust, G. Schaller, T. Beyer, D. Drasdo, Individual cell-based models of the spatial-temporal organization of multicellular systems – achievements and limitations, Cytometry A 69 (2006) 704–710.

[2] P. Murray, C. Edwards, M. Tindall, P. Maini, From a discrete to a continuum model of cell dynamics in one dimension, Phys. Rev. E 80 (2009) 031912.

[3] J. Fozard, H. Byrne, O. Jensen, J. King, Continuum approximations of individual-based models for epithelial monolayers, Math. Med. Biol. 27 (2010) 39–74.

[4] A. Middleton, C. Fleck, R. Grima, A continuum approximation to an off-lattice individual-cell based model of cell migration and adhesion, J. Theor. Biol. 359 (2014) 220–232.

[5] Y. Kim, M. Stolarska, H. Othmer, A hybrid model for tumor spheroid growth *in vitro* I: theoretical development and early results, Math. Mod. Meth. Appl. Sci. 17 (2007) 1773–1798.

[6] J. Jeon, V. Quaranta, P. Cummings, An off-lattice hybrid discrete-continuum model of tumor growth and invasion, Biophys. J. 98 (2010) 37–47.

[7] H. Hatzikirou, L. Brusch, C. Schaller, M. Simon, A. Deutsch, Prediction of traveling front behavior in a lattice-gas cellular automaton model for tumor invasion, Comput. Math. Appl. 59 (2010) 2326–2339.

[8] Y. Jiao, S. Torquato, Emergent behaviors from a cellular automaton model for invasive tumor growth in heterogeneous microenvironments, PLoS Ccomput. Biol. 7 (2011) e1002314.

[9] F. Graner, J. Glazier, Simulation of biological cell sorting using a two-dimensional extended Potts model, Phys. Rev. Lett. 69 (1992) 2013–2016.

[10] S. Turner, J. Sherratt, Intercellular adhesion and cancer invasion: a discrete simulation using the extended Potts model, J. Theor. Biol. 216 (2002) 85–100.

[11] N. Ouchi, J. Glazier, J.-P. Rieu, A. Upadhyaya, Y. Sawada, Improving the realism of the cellular Potts model in simulations of biological cells, Physica A 329 (2003) 451–458.

[12] J. Galle, M. Loeffler, D. Drasdo, Modeling the effect of deregulated proliferation and apoptosis on the growth dynamics of epithelial cell populations *in vitro*, Biophys. J. 88 (2005) 62–75.

[13] A. Fletcher, J. Osborne, P. Maini, D. Gavaghan, Implementing vertex dynamics models of cell populations in biology within a consistent computational framework, Prog. Biophys. Mol. Biol. 113 (2013) 299–326.

[14] A. G. Fletcher, M. Osterfield, R. Baker, S. Shvartsman, Vertex models of epithelial morphogenesis, Biophys. J. 106 (2014) 2291–2304.

[15] D. Drasdo, S. Höhme, A single-cell-based model of tumor growth *in vitro*: monolayers and spheroids, Phys. Biol. 2 (2005) 133–147.

[16] A. Fletcher, C. Breward, S. Chapman, Mathematical modeling of monoclonal conversion in the colonic crypt, J. Theor. Biol. 300 (2012) 118–133.

[17] G. Mirams, A. Fletcher, P. Maini, H. Byrne, A theoretical investigation of the effect of proliferation and adhesion on monoclonal conversion in the colonic crypt, J. Theor. Biol. 312 (2012) 143–156.

[18] P. Buske, J. Galle, N. Barker, G. Aust, H. Clevers, M. Loeffler, A comprehensive model of the spatio-temporal stem cell and tissue organisation in the intestinal crypt, PLoS Comput. Biol. 7 (2011) e1001045.

[19] P. Macklin, M. Edgerton, A. Thompson, V. Cristini, Patient-calibrated agent-based modelling of ductal carcinoma *in situ* (DCIS), J. Theor. Biol. 301 (2012) 122–40.

[20] N. Chen, J. Glazier, J. Izaguirre, M. Alber, A parallel implementation of the Cellular Potts Model for simulation of cell-based morphogenesis, Comput. Phys. Commun. 176 (2007) 670–681.

[21] B. Youssef, G. Cheng, K. Zygourakis, P. Markenscoff, Parallel implementation of a cellular automaton modeling the growth of three-dimensional tissues, Int. J. High Perform. Comput. Appl. 21 (2007) 196–209.

[22] S. Alberts, M. Keenan, R. D'Souza, G. An, Data-parallel techniques for simulating a mega-scale agent-based model of systemic inflammatory response syndrome on graphics processing units, Simulation 88 (2011) 895–907.

[23] M. Holcombe, S. Adra, M. Bicak, S. Chin, S. Coakley, A. Graham, J. Green, C. Greenough, D. Jackson, M. Kiran, S. MacNeil, A. Maleki-Dizaji, P. McMinn, M. Pogson, R. Poole, E. Qwarnstrom, F. Ratnieks, M. Rolfe, R. Smallwood, T. Sun, D. Worth, Modelling complex biological systems using an agent-based approach, Integr. Biol. 4 (2012) 53–64.

[24] S. Coakley, M. Gheorghe, M. Holcombe, S. Chin, D. Worth, C. Greenough, Exploitation of high performance computing in the FLAME agent-based simulation framework, in: G. Min, L. Lefevre, J. Hu, L. Liu, L. Yang, S. Seelam (Eds.), IEEE 14th International Conference on High Performance Computing and Communications, 2012, pp. 538–545.

[25] T. Newman, Modeling multicellular systems using subcellular elements, Math. Biosci. Eng. 2 (2005) 613–624.

[26] S. Christley, B. Lee, X. Dai, Q. Nie, Integrative multicellular biological modeling: a case study of 3D epidermal development using GPU algorithms, BMC Syst. Biol. 4 (2010) 107.

[27] P. Richmond, D. Walker, S. Coakley, D. Romano, High performance cellular level agent-based simulation with FLAME for the GPU, Brief. Bioinform. 11 (2010) 334–347.

[28] G. Mirams, C. Arthurs, M. Bernabeu, R. Bordas, J. Cooper, A. Corrias, Y. Davit, S.-J. Dunn, A. Fletcher, D. Harvey, M. Marsh, J. Osborne, P. Pathmanathan, J. Pitt-Francis, J. Southern, N. Zemzemi, D. Gavaghan, Chaste: an open source C++ library for computational physiology and biology, PLoS Comput. Biol. 9 (2013) e1002970.

[29] W. Coffey, Y. Kalmykov, J. Waldron, The Langevin equation: with applications to stochastic problems in physics, chemistry, and electrical engineering, Vol. 14, World Scientific, 2004.

[30] J. Dallon, H. Othmer, How cellular movement determines the collective force generated by the *Dictyostelium discoideum* slug, J. Theor. Biol. 231 (2004) 203–222.

[31] S. Plimpton, Fast parallel algorithms for short-range molecular dynamics, J. Comput. Phys. 117 (1995) 1–19.

[32] W. Gropp, E. Lusk, A. Skjellum, Using MPI: portable parallel programming with the message-passing interface, Vol. 1, MIT press, 1999.

[33] B. Karlsson, Beyond the C++ standard library: an introduction to boost, Pearson Education, 2005.