Cluster Comput (2015) 18:1561–1579
DOI 10.1007/s10586-015-0455-7

A parallel cellular automata algorithm for the deterministic simulation of 3-D multicellular tissue growth

Belgacem Ben Youssef[1]

Received: 18 September 2013 / Revised: 1 November 2014 / Accepted: 6 April 2015 / Published online: 22 April 2015

Abstract Besides generating faster solutions, parallel computers can be used to solve bigger or more complex problems. In particular, they can be utilized to run simulations at finer resolutions and to model physical phenomena more realistically. We describe in this article the development of a parallel cellular automata algorithm used in the three-dimensional simulation of multicellular tissue growth. Computational models of this genre are becoming ever more important because they provide an alternative approach to continuous models and an ability to describe the dynamics of complex biological systems evolving in time. We report on the different components of the model where cellular automata is used to model different types of cell populations that execute persistent random walks on the computational grid, collide, aggregate, and proliferate until they reach confluence. We elaborate on the main issues encountered in the parallelization of the algorithm as well as its implementation on a parallel machine with a particular focus on maintaining determinism. By delaying the exchange of cells in the shared boundaries between neighboring processors till after all events related to cell division and motion are accounted for in a given time step, good parallel performance results are obtained on a high-performance cluster.

Keywords Parallel algorithm · 3-D simulation model · Determinism · Multicellular tissue growth · Cellular automata · Performance sweet spot · Cluster machine

✉ Belgacem Ben Youssef
bbenyoussef@ksu.edu.sa

[1] Department of Computer Engineering, College of Computer & Information Sciences, King Saud University, Riyadh 11543, Saudi Arabia

1 Introduction

The need for computers that are multiple times faster than today's most powerful machines continues to grow unabated year after year. The demand for such computational power arises in many multidisciplinary applications, including the design of better drugs, the forecasting of severe storms, and the modeling of ecological and biological systems, to name just a few. One of the ways for increasing the computational power of computers is to use faster and faster components. Until recently, improvements in this area have been extraordinary. In fact, for nearly thirty years, Moore's law has predicted a doubling of transistor capacity every 18–24 months resulting in an average annual increase in processor performance between 25 and 52 % [1]. However, due to the long memory latency, the decrease in available instruction-level parallelism, and the limitations imposed on power consumption, the increase in processor performance has slowed recently. This has brought about a "sea change" characterized by a switch from uniprocessors to multiprocessors [2]. As a result, the exploitation of parallelism is expected to have more and more significance in the future while users will have to "think in parallel" more than ever before.

Cellular automata (CA) are dynamic systems, in which both space and time are discrete, consisting of a number of identical cells in a regular lattice. They were originally introduced by John von Neumann and Stan Ulam as a possible idealization of biological systems with a particular purpose of modeling biological self-reproduction [3]. Each cell in the cellular space (array or grid) can be in a finite number of states. The next state of each cell is determined, at discrete time intervals, according to its current state, the current state of its neighboring cells, and a next-state transition rule or function. Cellular automata provide a computationally

proficient technique for analyzing the collective properties of a network of interconnected cells. The use of cellular automata in modeling various systems including biological ones has a number of advantages that include the fact that CA are sufficiently simple to allow detailed mathematical analysis, yet complex enough to exhibit a wide variety of complicated phenomena. Models based on cellular automata provide an alternative approach involving discrete coordinates and variables to represent a complex dynamic system. In addition, algorithms based on cellular automata are also suitable for parallel processing [4,5].

The development of computational and simulation models for studying biocomplexity at the cell population and tissue level can provide powerful frameworks in this area, particularly by employing systems-based approaches [6,7]. These approaches consider cells as system components that migrate, proliferate and interact to generate the complex behavior observed in living systems [8,9]. However, employing systems-based approaches could lead to models with high complexity whose solution poses significant computational challenges [10–12]. The availability of computational models with *predictive* abilities could greatly speed up progress in this area by assisting scientists in predicting the dynamic response of cell populations to external stimuli, and by rapidly assessing the effect of various system parameters on the overall tissue growth rates. Computer simulations can thus be used to shorten the development stage by allowing researchers to quickly screen many alternatives and choose only the most promising ones for laboratory experimentation.

Our previous work in [13–15] showed that the simulations of realizable multicellular tissue objects is a computationally demanding task that requires small time steps to accurately describe the dynamics of multiple cell populations and long times to complete them. In addition to the size of the cellular array, several input parameters affect the execution time needed to run a simulation of this type. Some of these include initial cell seeding density, cell migration speed, and cell division time. For instance, we estimate that these factors, when combined together, yield a serial execution time of over 200 h for the simulation of tissue growth of 1 cm^3 in size. Based on the average size of the area of a mammalian cell at confluence, this represents a cellular space of 1000^3 computational sites, where it is assumed that each site has a side equal to 10 μm [15]. Such an outcome points to the need for using parallel computing systems in order to reduce the time to obtain simulation results. This article builds on this work by considering the parallelization of a three-dimensional computational and stochastic model for multicellular tissue growth using cellular automata while accounting for mammalian cell migration, division, collision, and aggregation. In the next section, we present some related work in this area. Afterwards, we describe the computational model and present the sequential algorithm. We then discuss different aspects related to its parallelization including the issue of maintaining determinism. After describing the parallel algorithm, we present the obtained performance results on a cluster. Finally, we provide our conclusion and future directions for our research. With this work, our overall contributions are twofold:

1. The development of a parallel algorithm using cellular automata to model tissue growth comprised of multiple cell populations, each with its own division and migration characteristics.
2. The implementation of the above parallel algorithm on a cluster machine while maintaining determinism and efficiency in terms of simulation and performance results, respectively.

2 Related work

Various modeling approaches have been used to simulate the population dynamics of proliferating cells. These models can be classified as: *deterministic*, *stochastic*, and based on *cellular automata* or *agents*. Deterministic models, such as the ones developed by Frame and Hu in [16] and Cherry and Papoutsakis in [17], provide insight into simple cell population dynamics. Such models may be useful in fitting specific quantitative results; but they give little or no topological information of the cell colonies before confluence or provide means of interpreting the parameters in terms of the biological processes involved.

Lim and Davies developed a stochastic two-dimensional model based on a matrix of irregular polygons and using the Voronoi tessellation technique to address the issue of cell topology [18]. While this model accounted for the formation and merging of cell colonies, it made some restrictive assumptions on cell interactions and did not address cell motility. Ruaan, Tsai, and Tsao proposed another stochastic model for the simulation of density-dependent growth of anchorage-dependent cells on flat surfaces [19]. Their model included the effects of cell motion and considered that cell sizes varied with time.

A two-dimensional model based on cellular automata was developed by Zygourakis, Bizios, and Markenscoff [20]. The model allows for contact inhibition during the proliferation process. Using the cellular automata concept, Hawboldt, Kalogerakis, and Behie [21] as well as Forestell, Milne, and Behie [22] modeled contact-inhibited synchronous cell growth on microcarriers. Both of these models were two dimensional and did not account for cell motion. Moreover, the assumption of uniform doubling time of cell populations was somewhat unrealistic in cell proliferation phenomena. Later, Lee et al. showed the importance of cell motility and cell-cell interactions in describing the cell proliferation

Table 1 Overview of deterministic, stochastic, and cellular automata models used in the simulation of cell proliferation dynamics and tissue growth

Author(s)	Model type	Main features	Limitations
Frame and Hu [16]	Deterministic	Growth rate based on cell density	All cells assumed to be equally contact-inhibited
Cherry and Papoutsakis [17]	Deterministic	Perimeter cell growth in cell colonies	Assumption of circular cell colonies only
			No colony mergings
Lim and Davies [18]	Stochastic	Random cell division	No cell motion
		Cells represented as irregular polygons	Restricted cell-cell interactions
Ruann et al. [19]	Stochastic	Density-dependent growth	Restricted cell motion
		Cell motility	Simplified cell division
Zygourakis et al. [20]	Cellular automata	Contact-inhibited cell proliferation	No cell motion
Forestell et al. [22]	Cellular automata	Cell growth on micro-carriers	Restricted number of cells and their neighbors
Hawboldt et al. [21]	Cellular automata	Cell growth for any	No cell motion
		cell line or microcarrier	Synchronous growth
Lee et al. [23,24]	Cellular automata	Cell motion and division	One cell per square (2D)
Kansal et al. [26]	Cellular automata	Tumor growth	No cell motion
		3-D model	No tracking of individual cells
Chang et al. [25]	Cellular automata	3-D model	No cell motion
		Cell division and death	No contact inhibition
Ben Youssef [15]	Cellular automata	3-D model	Single cell type
		Cell motion, division, and collision	
Cickovski et al. [27]	Cellular automata	Morphogenesis	No cell motion
		Multiple cell types	
Ben Youssef and Tang [13]	Cellular automata	Multiple cell types	Cell size does not vary
		Formation of multicelluar aggregates in 3D	

rates [23]. This work was succeeded by another model that described the locomotion of migrating endothelial cells in two dimensions [24].

Chang and his team developed a 3-D cellular automata based model to describe the growth of microbial unit cells [25]. This model considered the effects of bacterial cell division and cell death. Other models based on cellular automata have also been used to solve more specific modeling problems. For instance, Kansal et al. developed a model to simulate brain tumor growth dynamics [26]. Their model utilizes a few automaton cells to represent thousands of real cells, thus reducing the computational time requirements of the model while limiting its ability to track individual cells in the cellular space. Another cellular automata model was used by Cickovski et al. in [27] as a framework to simulate morphogenesis. This model used a hybrid approach to simulate the growth of an avian limb. The cellular automaton governed cell interactions while reaction-diffusion equation solvers were used to determine the concentration levels of surrounding chemicals. In Table 1, we present an overview of these deterministic, stochastic, and CA models while highlighting their main features and limitations.

Our extended three-dimensional cellular automata model represents a further refinement based on its inclusion of multiple cell types and its coverage of cell motion, division, collision, and aggregation. In particular, the model allows us to quickly evaluate the relative effect of many system parameters on the tissue growth rates including the initial density of seed cells and their spatial distribution as well as predict the time required for tissue growth given the properties of each cell population. Since natural tissues are multicellular and have a specific three-dimensional architecture, the simulation of tissue growth consisting of more than one type of cells becomes paramount. Further, because these multiple types of cells tend to organize themselves into very specific spatial patterns, the discrete cellular automata approach may be considered to be ideally suited to treat such problems with complicated geometry. Models with a predictive capability

are also suitable for the visual and quantitative exploration of a diverse range of testable hypotheses and "what-if" query scenarios, thus providing a basis for a rational design approach [28]. This makes them a necessary prerequisite for developing systems control strategies for biotechnological processes involving the proliferation and growth of contact-inhibited cells [29].

While our main focus here is on CA-based models, there also exist a number of agent-based lattice-free models to simulate tissue growth [30–32]. Agent-based models (ABMs) can be thought of as generalizations of CA models, where a number of individual and autonomous constituent entities (known as *agents*) interact locally in order to create a higher level, group behavior [10]. These models apply the dynamics of cell proliferation and death to describe tissue pattern formation and growth. Other related models are suitable for describing the locomotion of a fixed number of cells where cells move relatively slowly with respect to other processes like the diffusion of soluble substances [33,34]. Additional ABMs employ feedback mechanisms between cells and the substrate to model cells entering and leaving the tissue and to establish homeostasis in such systems [35]. Some of the agent-based models use regular triangulation to generate the neighborhood topology for the cells, thus allowing for a continuous representation of cell sizes and locations in contrast to grid-based models [36]. Others utilize multiscale approaches to model collective phenomena in multicellular assemblies [37]. The reader is referred to the recent work by Hwang et al., which reviews a number of rule-based modeling techniques of multicellular biological systems using, among others, agent-based models [10].

3 Engineering 3-D bioartificial tissues

Each year, millions of surgical procedures are performed to relieve patients who are affected by tissue loss, due to burns, injuries, or organ failure. Operations treating patients using tissue reconstruction and organ transplantation have been highly successful. However, the number of patients treated by these therapies is small due to the limited number of donors available [38]. The growth of three-dimensional tissue with proper structure and function is the main goal of tissue engineering. Tissue engineers draw on the knowledge gained in the fields of biology, biochemistry, engineering, and the medical sciences to develop bioartificial implants or to induce tissue remodeling in order to replace, repair or enhance the function of a particular tissue or organ [18]. The three-dimensional structure of natural tissues is supported by an *extracellular matrix* (ECM) that often has the form of a three-dimensional network of cross-linked protein strands. In addition to determining the mechanical properties of a tissue, the ECM plays many important roles in tissue development. Biochemical and biophysical signals from the ECM modulate fundamental cellular activities, including adhesion, migration, proliferation, as well as differentiation and programmed cell death [11,20]. For example, extracellular matrices may be used to promote wound healing, a serious problem with patients suffering from many debilitating diseases [8]. As shown in Fig. 1, the tissue engineering approach to wound healing consists of the following steps:

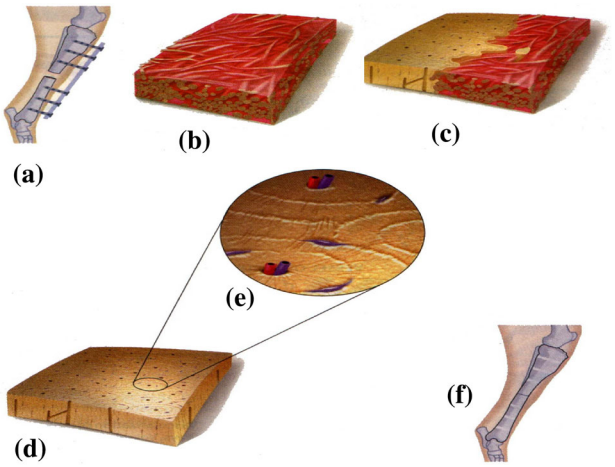

Fig. 1 New bone growth: a dog leg bone with a missing section is held in place with braces (*a*). A polymer scaffold primed with bone-growth-promoting proteins (*b*) fills the gap. The scaffold is slowly infiltrated by new bone (*c*). Ultimately, the degradable scaffold is completely replaced by bone (*d*). The new tissue (*e*) has its own blood supply (red and blue vessels). The leg bone has healed (*f*)

1. A biocompatible matrix is used to fill the defect (wound).
2. The scaffold may contain bioactive agents (growth factors) that induce neighboring cells to migrate into the scaffold, proliferate and produce their own extracellular matrix.
3. This process continues until the wound heals.

4 Computational model

Tissue regeneration is a highly dynamic process. When cells are seeded in a 3-D scaffold, they migrate in all directions, interact with each other and proliferate until they completely fill the space available to them. This assumes that enough nutrients are always available to sustain cell growth everywhere in the interior of the scaffold. To model this dynamic process, we consider cellular automata consisting of three-dimensional grids with N^3 total cubic computational sites [3,39]. Each site is a *finite automaton* that can exist in one of a finite number of states at each time interval that is interacting with its six immediate neighbors as shown in Fig. 2. That is, a site may be either:

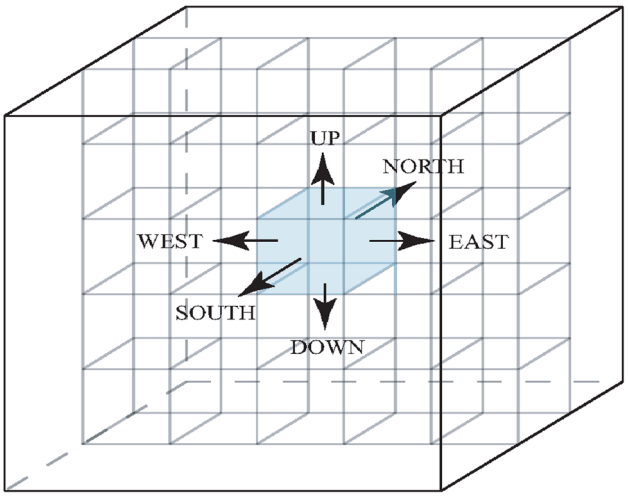

Fig. 2 A cell in the cellular automaton interacts with its six immediate neighbors. This is known as the von Neumann neighborhood in three dimensions

- empty and available for a cell to move in, or
- occupied by a cell, which is at a given point in its mitotic cycle and moves in a certain direction. No other cell can move or divide into an already occupied site.

Proliferating cells execute *persistent random walks* in space [40,41]. This process consists of the following stages:

1. Each cell in the population moves in one direction for a certain period of time (*persistence*). At the end of this interval, the cell stops and turns to continue its migration in another direction. The persistence is a random variable whose density function can be determined experimentally.
2. When two cells *collide*, they stop for a short period of time before resuming their migration to move away from each other.
3. At the end of its cycle, a cell stops to divide into two daughter cells.
4. This process is repeated until the cell population has completely filled the scaffold or until the cells cannot migrate and divide any further [13].

To simulate these dynamics, the state $x_j()$ of each cellular automaton takes values from a set of integer numbers that code all the required information about the cell type, its migration speed, the direction of movement, and the time remaining until the next direction change and the next cell division. Our model also considers cell *division time* as a random variable whose probability density function can be obtained experimentally using the procedure described by Lee and coworkers [24]. Hence, every automaton has its state evolving at discrete time steps Δt through interactions with neighboring automata. Let us consider the j-th automaton that contains a cell at time t_r. Its state $x_j(r)$ is specified by the following numbers:

1. Cell type $k_{t,j}$: For each cell population, this is a unique identifier. The number of modeled cell populations is based on the number of digits used to represent $k_{t,j}$. Using a single digit, up to nine different cell populations can be simulated with each population having its own division and migratory parameters.
2. Migration index m_j ($m_j \in \{0, 1, 2, 3, 4, 5, 6, 7\}$): If $m_j = 1, 2, \ldots, 6$, then the cell is migrating in one of the six directions (east, north, west, south, up and down). If $m_j = 0$, the cell is stationary. If $m_j = 7$, the cell is in the aggregation state.
3. Division counter $k_{d,j}$: The time that must elapse before the cell divides is equal to $t_d = k_{d,j}\Delta t$. For each iteration, this counter is decremented by one and the cell divides when $k_{d,j} = 0$.
4. Persistence counter $k_{p,j}$: The time that must elapse before the cell changes its direction of movement is equal to $t_p = k_{p,j}\Delta t$. For each iteration, this counter is decremented by one and the cell turns when $k_{p,j} = 0$.

For every automaton $j (1 \leq j \leq N^3)$, the application of these rules defines a local transition function specifying the state $x_j(r+1)$ of the automaton at $t_{(r+1)}$ as a function of the states of its neighbors at t_r, that is

$$x_j(r+1) = f_j(x_j(r), x_{j+1}(r), x_{j+2}(r), \ldots, x_{j+6}(r)), \quad (1)$$

where $x_{j+1}(r), x_{j+2}(r), \ldots,$ and $x_{j+6}(r)$ are the states of the six neighbors of automaton j. The application of the local transition functions, $f_j(\ldots)$, to all the automata in a cellular space transforms a configuration $X(r) = [x_1(r), x_2(r), \ldots, x_{N^3}(r)]$ of the cellular automaton to another one $X(r+1)$ according to Eq. 1. Thus, a global transition function F acting on the entire array can be defined as follows:

$$X(r+1) = F(X(r)), \quad r = 0, 1, 2, \ldots \quad (2)$$

As a result, starting from an initial configuration $X(0)$, the cellular automaton follows a trajectory of configurations $X(1), X(2), \ldots, X(r), \ldots$ defined by the global transition function F. At each time level, the states of all the cells of an automaton are updated in parallel.

5 Sequential algorithm

5.1 Initial condition

The sites that will be occupied by cells at time t_0 are selected. The assignment of seed cells to the grid sites may be done

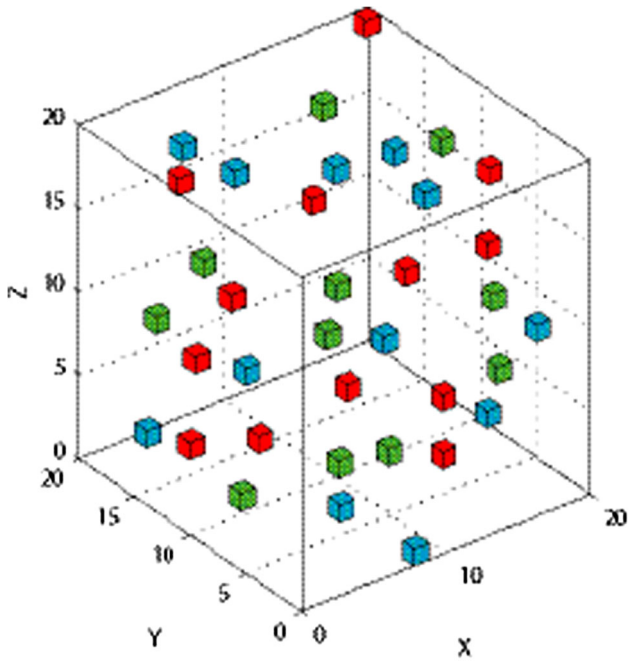

Fig. 3 An example of a uniform cell-seeding distribution displaying three cell types in a *mixed* seeding mode. A total of 40 cells in a $20 \times 20 \times 20$ cellular array are seeded, yielding a 0.5 % cell-seeding density

either randomly (using, for example, a uniform distribution like the one depicted in Fig. 3) or according to rules that emulate special cases of tissue regeneration like wound healing [8]. Afterwards, an initial state $x_j(0)$, at time t_0, is assigned to each occupied site j based on the population characteristics of that cell type. The migration index m_j is randomly selected, the value of the persistence counter $k_{p,j}$ is properly chosen, and the cell division counter $k_{d,j}$ is set according to the experimentally determined distribution of cell division times. As stated previously, the integer counters $k_{p,j}$ and $k_{d,j}$ will be decremented at every iteration and the cell will change its direction of movement or divide when $k_{p,j} = 0$ or $k_{d,j} = 0$, respectively.

5.2 Iterative operations

At each time step $t_r = t_{(r-1)} + \Delta t, r = 1, 2, \ldots$

1. Randomly select a computational site.
2. If this site contains a cell that is ready to divide, execute the *division routine* and go to step 5.
3. If this site contains a cell that is ready to change its direction of movement, execute the *direction change routine* and go to step 5.
4. Otherwise, try to move the cell to a neighboring site in the direction indicated by the migration index of its current state:
   a. If this site is free, *mark* it as the site that will contain the cell at the next time step and decrement the persistence and cell phase counters by one.
   b. If this site is occupied by a cell from a different cell type, we have a cell-cell collision. The cell will remain at the present site by entering the *stationary state* and will execute the *direction change* routine after a pre-specified number of iterations. Its cell phase counter is decremented by one.
   c. If this site is occupied by a cell from the same cell type, we have a cell-cell aggregation. The cell will remain at the present site by entering the *aggregation state* and will execute the *direction change* routine after a pre-specified number of iterations. Its cell phase counter is decremented by one.
5. Select another site and repeat steps 2–4 until all sites have been examined.
6. Update the states of all sites so that the locations of all cells are set for the next time step.

5.3 Division routine

1. Scan the neighborhood of the current site to determine if there are any free adjacent sites. If all adjacent sites are occupied, the cell will not divide. The cell phase counter gets a new value.
2. If there are free sites in the neighborhood, select one of these sites to place the second daughter cell. The other daughter cell will occupy the current location. The selection algorithm may assign either the same probability to each of the free neighbors of a cell or "bias" the division process by assigning higher probabilities to some neighbors.
3. Mark the selected site that will contain one of the daughter cells in the next time step. Once a site has been marked, no other cell can move in it during this iteration. Set the state of this site $x_j(r)$ by defining the migration index as well as the persistence and cell division counters.

5.4 Direction change routine

1. Scan the neighborhood of the current site to determine if there are any free adjacent sites. If all adjacent sites are occupied, the cell remains at the present site. The cell is also assigned a new persistence counter.
2. If there are free sites in the neighborhood, select one of these sites by using a random algorithm based on the experimentally determined state-transition probabilities.

3. Mark the selected site that will contain the cell in the next time step to prevent other cells from occupying it. Set the persistence counter to its appropriate initial value and decrement the cell phase counter by one.

6 Parallelization steps

The parallel algorithm we designed to simulate the dynamics of multicellular tissue growth was implemented on a distributed-memory cluster machine using the Message Passing Interface (MPI) [42,43]. Our discussion here will focus on the main issues we faced during the different steps comprising the parallelization task.

6.1 Application architecture

We begin by looking at the architecture of the application at hand. Our application falls in the category of *loosely synchronous* applications as categorized by Pancake [44]. Such applications exhibit certain characteristics where the amount of computation could vary from one partition and time step to another because it depends on the amount of useful data, which is proportional to the number of occupied sites in our case [44]. A single processor experiences different workloads from the early time steps to the later ones as cells divide and proliferate. The need to exchange data among processors necessitates that each processor be able to determine when the other nodes are ready for this exchange so that data not yet used are not overwritten. Between these exchange points, the different nodes proceed at their own rates.

6.2 Load balancing

Since the workload now varies both temporally and spatially, much effort must be spent to evenly distribute it among the nodes. In order to minimize overhead, we used a *static* and cell distribution-dependent *load-balancing* strategy whereby each node stays responsible for a fixed part of the cellular space. This is known as the *Eulerian* method [45]. Static methods are uncomplicated, but can have difficulty handling subsequent load imbalances. The major advantage of using static load balancing is that the entire overhead of the load-balancing process is incurred at compile time. This represents a one-time and fixed cost that results in more efficiency.

Our choice of a static load-balancing strategy was based on the fact that the behaviour of cells (their division and migration) is random. Since the seed cells were evenly and uniformly distributed throughout the global cellular array, we observed that the computational load fluctuations between neighboring sub-domains tend to average out, thus maintaining a load-balanced computation. This means that the number of cells leaving a sub-domain is counterbalanced by a nearly equal number of cells entering it. Moreover, the choice of a cell distribution-dependent load-balancing technique fulfilled the dual objective of not only conserving the required load balancing but also that of maintaining the efficiency of the parallel computation. Further, this choice helps us achieve solutions which are scalable, that is, solutions that should be efficient for a wide range of number of processors, with the goal of minimizing the overall execution time of the program, while minimizing the communication delays.

6.3 Domain decomposition and mapping

We used a seeding mode where cells were uniformly and randomly distributed in the cellular array as shown in Fig. 3. This is widely known as the most common seeding mode in tissue engineering applications [46]. This type of distribution is amenable to a slab decomposition that can be achieved by dividing the cellular array into equal partitions along the z dimension. As displayed in Fig. 4, the area of the boundary between any two sub-domains is equal to $N_x \times N_y$ sites. Hence, the maximum amount of data communicated from a sub-domain to its neighbors is $2 \times N_x \times N_y$ data elements at any given instant. To implement the slab decomposition, the sub-domains were logically mapped onto a one-dimensional processor grid representing a topology of a linear array. In our application, this topology reflects the logical communication pattern of the parallel processes. It also preserves the locality property of the algorithm, thus guaranteeing that all neighbors of a cell in the cellular space are stored either on the same processor or on a processor that is logically an immediate neighbor to it.

6.4 Handling cell movement and division

Performing cell movement and division in one step will lead to correctness problems. The solution is to use a "splitting" technique that consists of two aggregate steps: the first computes the next position of a cell or its daughter cell, while the second step is the one that actually moves/divides the cell and updates the cell state information [47]. The sequential implementation inherently uses this splitting technique so that there is no possibility of conflicts. To preserve the semantics of the serial algorithm, the parallel algorithm must communicate with the neighboring processors between the first and second step. If we do not communicate then, we lose correctness since this leads to having more than one cell occupy a given site, which is a violation of one of the rules of our cellular automaton.

6.5 Ensuring determinism

In the context of determinism, our objective is to obtain reproducible results for the same input data. This is not necessarily

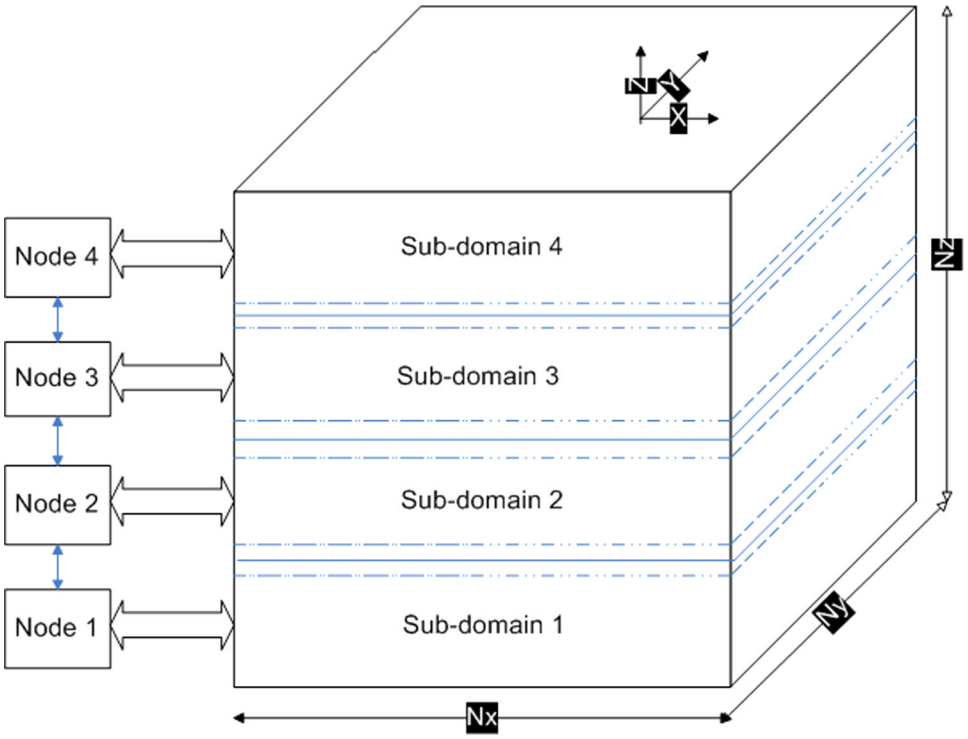

Fig. 4 The domain decomposition of the cellular grid on a cluster with P = 4 nodes (processors). The mapping shows how each node is assigned a fixed partition of the cellular space in the shape of a *slab*. The *dotted lines* denote shared boundaries with neighboring processors

fulfilled for a given algorithm, even if this algorithm is correct [48]. Deterministic results are paramount for the following reasons:

- *Verification* Results are often only useful if we have a way to *verify* them with different means. To get the same results whether they are obtained by a single processor, four processors, or by 32 processors is a strong argument for the verifiability of these results. Thus, determinism in this case supports correctness.
- *Comparison* We may want to compare the performance of different systems. Ensuring that similar results are obtained on these systems is needed to achieve this task. Thus, determinism in this case is an important condition that supports system evaluation.

The key to determinism is to define precise procedures for making decisions and to define a precise order to perform these procedures. We have to make decisions whenever there are several possible actions and we must choose one of these. In our application, for example, we have to decide where to move a cell or where to place a daughter cell after division, in case the cell in question has several empty sites in its neighborhood.

In a parallel algorithm, there are at least two factors that make determinism hard to ensure:

1. *The decomposition of the problem domain* This may depend on the number of nodes or the kind of load balancing done. Dynamic load-balancing strategies introduce significant complications in this regard that are due to the needed rebalancing of the workload on processors at run time.
2. *The hardware* SIMD architectures may have a central clock, making the hardware completely deterministic. However, this is usually not the case for high performance clusters. Every processing node in a cluster has its own clock, and there are always slight speed differences. For instance, if two nodes want to communicate with a third node at the same time, we cannot predict which node will be serviced first.

Because of such problems, it is hard to produce a stable parallel code that will always give the same results. To overcome these issues, we used a static load-balancing strategy that is dependent on the initial cell distribution topology as discussed previously. In addition, in our algorithm, initially cells are randomly seeded. When simulation starts, cells are then randomly chosen to perform predefined procedures and to reset the state of a site occupied by a living cell. Our serial algorithm uses often a pseudo-random number generator (PRNG) to effect these computations and allow the simulation of tissue growth to proceed according to the cellular automaton rules for cell division, migration, collision, and aggregation. In essence, the PRNG plays a key role in keeping the computations deterministic.

A parallel implementation of the PRNG on one node would be very inefficient and would constitute a bottleneck.

Instead, we parallelized the random number generator in a way that keeps our computations reproducible and deterministic, even if the order of the generation of the parallel subsequences by the different nodes is not fixed. The key for this is the parallelization strategy and, consequently, the seeds used. We employed the leaping strategy which interleaves the generation of the parallel subsequences of random numbers [49]. This interleaving allows us to obtain in a deterministic way *unique* parallel seeds for each of the parallel simulation runs based on a given seed for a serial simulation run. We discuss next how this approach was implemented.

6.5.1 *Parallelization of random number generation on a cluster*

Based on its randomness quality, its amenability to be parallelized in a way that ensures the reproducibility of simulation results, and the provision of parallel streams of random numbers that are adequately independent from one another, we chose to implement the multiplicative linear congruential method, first presented by D. H. Lehmer, with *carefully* chosen parameters [50]. The generator is defined as $x_{n+1} = ax_n \mod m$,where x_n is the nth member of the sequence of random numbers before normalization, m is a large prime number ($m = 2^{31} - 1$), and a is an integer ranging from 2 to $m - 1$, with the specific property that it is a primitive root of m to ensure a full length period ($a = 62,089,911$) [51]. This sequence must be initialized by choosing an initial integer value, $x_1 \in \{1, 2, \ldots, m - 1\}$, called the *seed*. By a simple manipulation of the generating equation, we can compute the $(n + k)$th member of the sequence in terms of the nth as follows:

$$x_{n+k} = Ax_n \mod m, \text{where} A = a^k. \tag{3}$$

We assume that our high performance cluster consists of P processors, or nodes, interconnected by some communication network. The idea is to have each processor compute its random numbers using the previous equation with $k = P$. Since the value of P is known for each parallel simulation run, the quantity A can be computed once and stored. We start this process by giving the processors a staggered start to prevent their respective subsequences from overlapping.

Let y_1 denote the seed of the sequential algorithm ($y_1 = x_1 = 123$). We set the seeds in each processor of the cluster in the following way, where a subscript denotes the position in the random number sequence and a superscript denotes a processing node,

$$\begin{cases} x_1^{(1)} = y_1 \\ x_1^{(2)} = ay_1 \mod m = y_2 \\ x_1^{(3)} = ay_2 \mod m = a^2 y_1 \mod m = y_3 \\ \vdots \\ x_1^{(P)} = a^{P-1} y_1 \mod m = y_P. \end{cases}$$

The above set of formulae achieves the previously mentioned staggered start and results in *unique* seeds for each parallel subsequence. Each node in the cluster then uses Eq. 3 to calculate the next member of its sequence, where k is now replaced with P. This process is repeated for the subsequent random number calculations and is exemplified below for node i, $1 \le i \le P$:

$$\begin{cases} x_2^{(i)} = y_{i+P} = Ay_i \mod m = (a^P \mod m) x_1^{(i)} \mod m \\ x_3^{(i)} = y_{i+2P} = a^P y_{i+P} \mod m = (a^P \mod m) x_2^{(i)} \mod m \\ \vdots \\ x_j^{(i)} = y_{i+(j-1)P} = a^P y_{i+(j-2)P} \mod m = (a^P \mod m) x_{(j-1)}^{(i)} \mod m. \end{cases}$$

The adoption of a single PRNG allows us to have a single copy residing in the local memory of each processing node. The parallel generation of random numbers then proceeds in an interleaved fashion to avoid any memory conflicts. Based on the asynchronous processing of computational sites in the cellular space, each node locally controls the rate of production and consumption of random numbers in its subsequence. This has the effect of nodes jumping ahead one another while computing their respective next random numbers in their subsequences [49].

The new multiplier for the parallel generation of these subsequences is now given by $a^P \mod m$. Computing this multiplier efficiently may be achieved by exploiting the associativity property of the modulo operation with respect to multiplication using a divide-and-conquer strategy. Further, we note that choosing a single PRNG gives us the flexibility to properly select its parameters and to use knowledge gained thus far in determining and understanding its statistical properties [52].

7 Parallel algorithm

The goal of the developed parallel algorithm is to reduce the amount of communication between nodes by exchanging shared boundaries during a simulation time step *only* when a process has calculated the movement/division of all cells in the sub-domain and not each time a cell attempts to cross over to a neighboring sub-domain. Thus, for cells attempting

to cross over to a neighboring sub-domain, their inquiries are recorded and sent to the corresponding neighbor, only after all the cells of the sub-domain have been considered. The actual movement/division of a cell that crosses over to a neighboring sub-domain is performed after the exchange of the shared boundaries. Using the current processor as a reference point, we define the following terms and notations that will be used in describing the steps of the parallel algorithm:

- **Myself** Identifies the id number of the current processor (for example, id i).
- **Mypred** Identifies the id number of the predecessor processor using a logical linear array topology (for example, id $i-1$).
- **Mysucc** Identifies the id number of the successor processor using a logical linear array topology (for example, id $i+1$).
- **Volume Coverage** Represents the percentage of occupied sites. A value of 99.99 % is usually chosen. This is also known as the *confluence* parameter.
- **X(N_x, N_y, N_z)** The global cellular array containing all cells.
- **X(N_x, N_y, 0:n_z+1)** The part of the cellular array owned by the current processor (local sub-domain) including two ghost layers to accommodate shared boundaries with the two neighboring processors, where $n_z = N_z/P$.
- $\mathbf{M}_{crossing\_to\_mypred}$ A 2-D matrix containing the state information of all cells attempting to cross over from the bottom layer (layer 1) of current processor to the top layer (layer n_z) of predecessor processor.
- $\mathbf{M}_{crossing\_to\_mysucc}$ A 2-D matrix containing the state information of all cells attempting to cross over from the top layer (layer n_z) of current processor to the bottom layer (layer 1) of successor processor.
- $\mathbf{M}_{crossing\_from\_mypred}$ A 2-D matrix containing the state information of all cells attempting to cross over from the top layer (layer n_z) of predecessor processor to the bottom layer (layer 1) of current processor.
- $\mathbf{M}_{crossing\_from\_mysucc}$ A 2-D matrix containing the state information of all cells attempting to cross over from the bottom layer (layer 1) of successor processor to the top layer (layer n_z) of current processor.
- $\mathbf{M}_{rejected\_to\_mypred}$ A 2-D matrix containing the position and state information of all cells rejected by current processor and going back to predecessor processor.
- $\mathbf{M}_{rejected\_to\_mysucc}$ A 2-D matrix containing the position and state information of all cells rejected by current processor and going back to successor processor.
- $\mathbf{M}_{rejected\_by\_mypred}$ A 2-D matrix containing the position and state information of all cells rejected by the predecessor processor and going back to current processor.
- $\mathbf{M}_{rejected\_by\_mysucc}$ A 2-D matrix containing the position and state information of all cells rejected by the successor processor and going back to current processor.
- **Layer** 0 Represents layer n_z of the predecessor processor.
- **Layer** n_z+1 Represents layer 1 of the successor processor.

We describe in the parallel algorithm the actions of an even-numbered process P_{2i} during the k^{th} simulation time step. Unless otherwise specified, "send" and "receive" in the pseudo-code mean "MPI_ISEND" and "MPI_IRECV", respectively, which represent a non-blocking mode of inter-processor communication [43]. The main operations of this process are performed in the following order:

1. Calculation of cell movement/division.
2. Execution of cell movement/division.
3. Sending a message.
4. Receiving a message.
5. Updating the sub-domain according to the information provided by the received message.

Parallel Algorithm: Pseudo code showing the actions performed by an even-numbered process P_{2i} during the k^{th} simulation time step.

Inputs: $X(N_x, N_y, 0{:}n_z{+}1)$ at the end of $(k\text{-}1)^{st}$ time step, myself, mypred, mysucc, and volume coverage.

Output: $X(N_x, N_y, 0{:}n_z{+}1)$ at the end of k^{th} time step.

```
While the specified volume coverage has not been reached Do
|   While not all cells in the sub-domain have been considered Do
|   |   Randomly select the next cell in the sub-domain;
|   |   Calculate cell movement and division;
|   |   If the cell is not crossing over to a neighboring sub-domain Then
|   |   |   Execute its movement and division;
|   |   Else
|   |   |   Record the cell's current position and the calculated new cell
|   |   |   state: 1) in a matrix M_crossing_to_mypred for cells attempting to cross
|   |   |   over from layer 1 of myself to layer n_z of mypred and 2) in a
|   |   |   matrix M_crossing_to_mysucc for cells attempting to cross from layer n_z of
|   |   |   myself to layer 1 of mysucc;
|   |   EndIf
|   |   Send M_crossing_to_mypred to mypred and M_crossing_to_mysucc to mysucc;
|   |   Receive M_crossing_from_mypred from mypred and M_crossing_from_mysucc from mysucc;
|   |   For each cell recorded in M_crossing_from_mypred and M_crossing_from_mysucc Do
|   |   |   Decide whether a cell that crosses over from a neighbor will be
|   |   |   accepted (into an empty site of the sub-domain) or rejected due to
|   |   |   having more than one cell moving to a given site;
|   |   |   Record the position of a rejected cell in a matrix M_rejected_to_mypred for
|   |   |   cells going back to mypred and in a matrix M_rejected_to_mysucc for cells
|   |   |   going back to mysucc;
|   |   EndDo
|   |   Send M_rejected_to_mypred to mypred and M_rejected_to_mysucc to mysucc;
|   |   Receive M_rejected_by_mypred from mypred and M_rejected_by_mysucc from mysucc;
|   |   For each cell recorded in M_rejected_by_mypred and M_rejected_by_mysucc Do
|   |   |   Recalculate cell movement and division from the cell's original
|   |   |   position;
|   |   |   Execute cell movement and division;
|   |   EndDo
|   EndDo
|   Send layer 1 to mypred and layer n_z to mysucc;
|   Receive layer 0 from mypred and layer n_z+1 from mysucc;
|   Update the time step of the simulation;
EndDo
```

In order to avoid *deadlock*, the odd-numbered and even-numbered processes would execute these operations in a different order. Thus, an odd-numbered process P_{2i+1} executes these operations during the same k^{th} simulation time step in the following order:

1. Receiving a message.
2. Updating the sub-domain according to the information provided by the received message.
3. Calculation of cell movement/division.
4. Execution of cell movement/division.
5. Sending a message.

We note, from the above, that the even-numbered processes have an advantage over the odd-numbered processes when they compete with them for an empty site in the shared boundaries, as they can execute cell movement/division first. To counterbalance the effect of this computational artifact, the previously described order of the main operations for the even and odd-numbered processes is switched during the next time step, i.e., during the $(k + 1)^{th}$ simulation time step the even-numbered processes will execute these main operations in the order that the odd-numbered processes executed them during the k^{th} simulation time step. This process is repeated until the desired *confluence* value is reached.

7.1 Sample of simulation results

We present here only a sample of the sequential and parallel simulation results. These results were obtained using a $200 \times 200 \times 200$ cellular array having a uniform cell distribution with an initial cell-seeding density of 0.5 % using a *mixed* mode and a confluence parameter of 99.99 %. Two cell populations were employed in these runs, one moving at a speed of 10 μm/h while the other at a slower pace of 1 μm/h. We define the *cell heterogeneity* measure, H, as the ratio of the number of seed cells from population 1 to that from population 2, where cell population 1 contains all cells moving at the faster speed (10 μm/h). For the parallel simulations, the slab decomposition technique with eight processors was utilized in this case. The simulation results display the effect of varying the ratio H on volume coverage, shown here as the cell volume fraction, $k(r)$, which is equal to the number of occupied sites divided by the size of the cellular space at time step t_r. Both Figs. 5 and 6 show that increasing the value of H yields a decrease in the time to reach complete volume coverage. This is because for larger values of H, the population of faster moving cells dominates the proliferation process. In a *mixed* mode, such faster moving cells use their speed to spread out in the cellular space while seeking available empty sites, thus preventing the formation of cell colonies while allowing for confluence to be reached sooner. We further observe that for larger values of $H (H > 5)$, the results

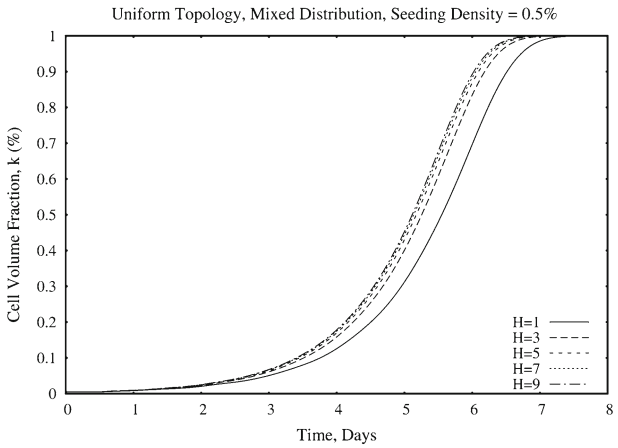

Fig. 5 Sequential simulation results showing the effect of varying the cell heterogeneity ratio, H, on the cell volume fraction

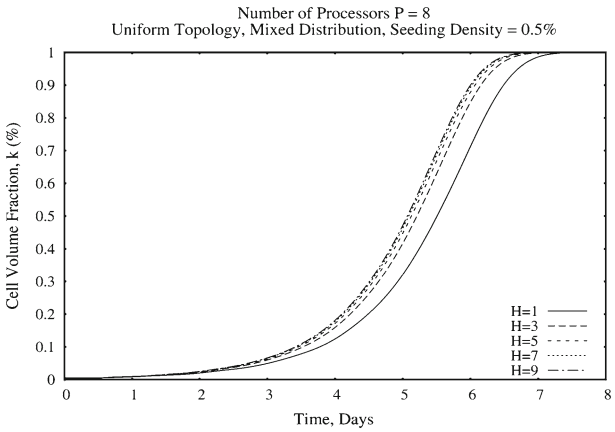

Fig. 6 Parallel simulation results showing the effect of varying the cell heterogeneity ratio, H, on the cell volume fraction

of the temporal evolution of cell volume fraction become indistinguishable from one another. This may suggest that scientists should limit their experimentations in this context to values of $H \leq 5$. These two figures also show how close the parallel simulation results are to the sequential ones.

8 Parallel performance

8.1 Computing platform and experimental conditions

The computing platform used to implement the parallel algorithm is a high-performance computing cluster, under the name of *Nebula*, located in the InfoNet Media Centre at Simon Fraser University. The cluster consists of 128 interconnected nodes using Gigabit Ethernet. Each node is comprised of an Intel P4 3.0-GHz processor with 2 GB of RAM. These nodes run the Gentoo Linux operating system with a GCC compiler version 4.4.3 and a LAM-MPI version

7.1.4. MPI is becoming a *de facto standard* to implement message passing on distributed-memory machines while employing the Single Program Multiple Data (SPMD) programming model [53].

Our performance results were obtained by running both the sequential and parallel algorithms on the cluster. The following experimental conditions were observed in order to generate consistent timing results for our simulation experiments:

- *Use the best available resources* While access to the cluster is allowed uninterrupted for researchers, particular care was taken to run our simulations when the load on the cluster was the smallest (most experiments were run between 2:00am and 8:00am). The Portable Batch System (PBS) server handles the task of job management by allocating nodes to the various jobs in the queue, thereby freeing the user from processor management issues.
- *Use the best compiler options* We compiled our programs using `mpic++ -O2 -march=pentium4 parallel_program.cpp`—for the parallel program, and `g++ -O2 -march=pentium4 serial_program.cpp`—for the serial program. The *-O2* option provides the highest optimization level in the *g++* compiler without introducing errors while the *-march* option instructs the compiler to produce processor-efficient code.
- *Use high-level, portable C++ code* Except for the standard I/O, timer libraries, and MPI, no other libraries or assembly code were used. We also did not rely on the system libraries to generate pseudo-random numbers. Instead, we implemented our own PRNG based on the multiplicative linear congruential method with properly chosen parameters, both sequentially and in parallel, as discussed previously [54].
- *Measure both execution and communication times* We measured the execution time of the sequential program using the *clock()* function, which is monitored at a resolution of one microsecond. This represents the fastest total execution time (in seconds) of the sequential program running on one node of the cluster. For parallel runs, we measured both the execution and communication times using the MPI function *MPI_Wtime()*. To ensure consistent performance results, each program was executed twelve times, and the best time result was reported. This is because it corresponds to the simulation run experiencing the least interference from the operating system. Further, particular care was taken to run these simulations when the load on the system is the smallest to minimize interference from other user tasks. For this purpose, we define the *parallel execution time* to be the fastest total execution time, in seconds, of the parallel program running on P nodes, including communication time. The latter metric is defined as the total time spent by the parallel program running on P nodes performing communication operations. Such operations include point-to-point, collective, and aggregated communications [43].

8.2 Performance results and discussion

The time complexity of our implementation of the sequential algorithm is of the order of $O(N^3)$. In addition, our implementation of the parallel algorithm based on the slab decomposition has a time complexity of the order of $O(N^3/P)$, using P processors. As mentioned above, the sequential execution time is obtained by running the serial algorithm on a single processor. During our simulation experiments, we were limited by the available memory capacity per node. For instance, the largest cellular array size for the sequential runs was $330 \times 330 \times 330$. We present herein performance results that were obtained for a uniform cell distribution with an initial cell-seeding density of 0.5 % using a *mixed* mode. Two cell populations with equal initial numbers were employed in these runs (that is, $H = 1$), one moving at a speed of 10 μm/h while the other at a slower pace of 1 μm/h. For each cellular array size, we varied the number of processors P from 2 to 50 and for each selected number of processors in this range, we varied the size of the cellular array such that $N \in \{150, 200, 250, 300, 330\}$, with $N = N_x = N_y = N_z$. The measured execution times for different cellular array sizes and numbers of processors are shown in Table 2.

8.2.1 Parallel speedup and efficiency

Parallel speedup and efficiency, denoted by S and E respectively, are two of the most commonly accepted performance measures of an application running on a parallel computer system. The speedup is equal to the sequential time divided by the parallel execution time, for specific values of P and N. In turn, the efficiency is set equal to the speedup divided by the number of processors [55]. Using the execution time data provided in Table 2, the speedup and efficiency values were computed and then plotted. They are presented in Figs. 7, 8, 9, and 10 to show their comparison for various numbers of processors and cellular array sizes.

In particular, the performance results displayed in Figs. 7 and 8 show that for a fixed cellular array size and as the number of processors increases, the speedup values mostly increase while efficiency decreases throughout. There are two observed exceptions in regards to this outcome whereby speedup starts to decrease at some intermediate numbers of processors. The first exception is in the case of $N = 150$ and $P > 20$ processors while the second one became manifest for $N = 200$ and $P > 25$ processors. Overall, this increase in speedup values is due to the fact that increasing the number of processors yields, for most cases, smaller execution

Table 2 Execution times, in seconds, of the parallel algorithm on a cluster for various cellular array sizes and different numbers of processors

Grid size (N × N × N)	Number of processors							
	1	2	4	8	10	20	25	50
150 × 150 × 150	2124	1441	861	570	538	**539**	**571**	**637**
200 × 200 × 200	5413	3568	1942	1181	1047	838	816	**872**
250 × 250 × 250	11,382	7389	3911	2292	1925	1361	1255	1124
300 × 300 × 300	21,211	13,199	6884	3838	3291	2029	1833	1497
330 × 330 × 330	29,156	17,873	9255	5072	4266	2615	2298	1765

The values shown in bold type indicate a shift from an increase to a decrease in performance of the parallel algorithm

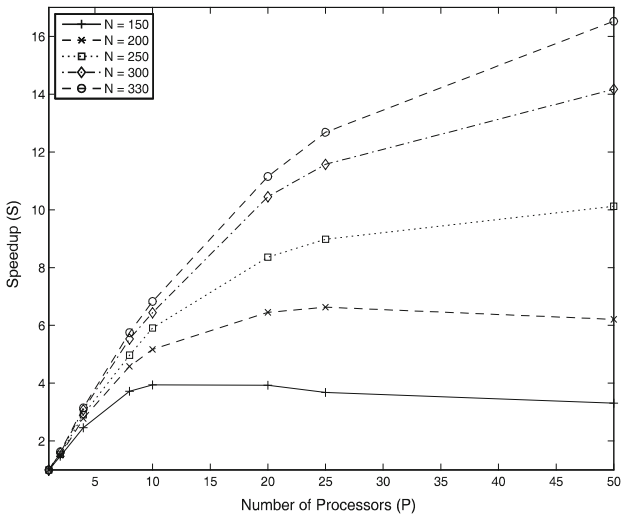

Fig. 7 Comparison of speedup *curves* for various cellular array sizes, N, as a function of the number of processors (P)

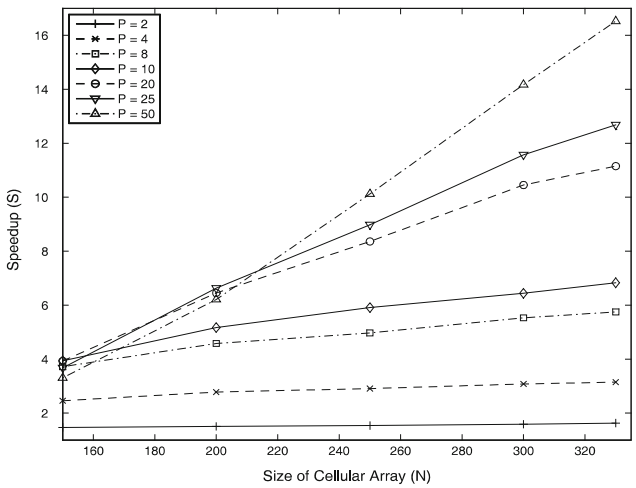

Fig. 9 Comparison of speedup *curves* for various numbers of processors, P, as a function of the size of the cellular array (N)

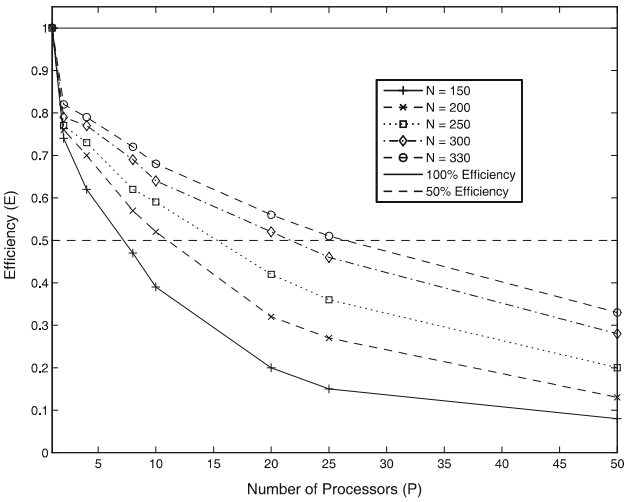

Fig. 8 Comparison of efficiency *curves* for various cellular array sizes, N, as a function of the number of processors (P)

times. On the other hand, the decrease in efficiency is due to the fact that increasing the number of processors increases the communication time for a fixed problem size (N) and, thus increases its related overhead.

In addition, the results exhibited in Figs. 9 and 10 show that for a given number of processors and as the cellular array size increases, both the speedup and efficiency values increase. This is due to the fact that as the size of the cellular array increases, more computational sites in each sub-domain are available for processing, thus resulting in more data parallelism being available in each node. This also means that more useful computational work is being accomplished by each processor yielding a larger ratio of execution time over communication time and leading to a coarser granularity of communication.

8.2.2 Communication overhead

The incurred communication overhead by our simulation model can be mostly attributed to the communication requirements of the parallel algorithm. Such requirements are often proportional to the amount of work a node has to perform near the boundaries of its sub-domain. This is related to the work the node must communicate to its neighbors. Hence,

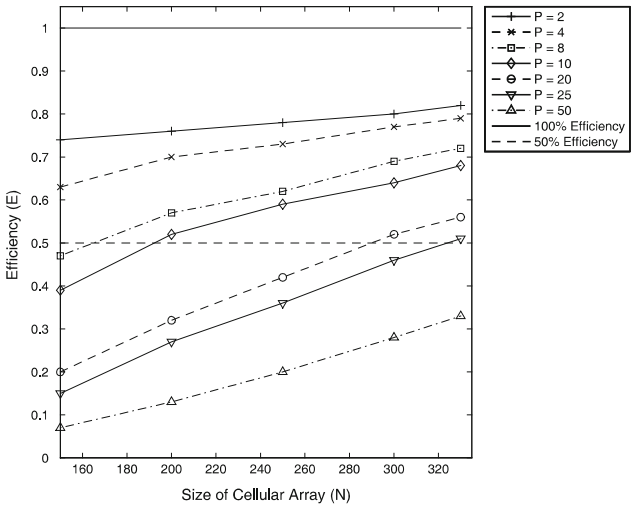

Fig. 10 Comparison of efficiency *curves* for various numbers of processors, P, as a function of the size of the cellular array (N)

a node's communication needs are proportional to the product of the work density and the surface of its slice of the sub-domain. In our tissue growth simulation model, the communication needs are determined by the way we process and update cells in the shared boundaries between neighboring sub-domains. This requires both an exchange of boundary sites and an exchange of computed cell divisions and movements to shared sites after *all* cells in a particular sub-domain are considered. Such exchanges between neighboring nodes are decomposed into two phases: communication with the top neighbor followed by communication with the one to the bottom. We use an algorithm where all boundary sites from a neighbor are brought in one message. *Coalescing* of communication in this manner reduces the number of messages and improves performance [53].

There are three types of MPI communication operations that account for the total communication time. They include point-to-point communication, collective communication, and aggregated computation. Examples of primitives that implement these operations include: send and receive, reduction, and barrier, respectively. In a point-to-point communication, only two nodes, the sender and the receiver, are involved. In a collective communication operation, tasks in a group send messages to one another, and the time is a function of both the message length and the number of nodes. For instance, in a broadcast operation, a single node sends an m-byte message to the remaining $P - 1$ nodes. In an aggregated computation, tasks in a group synchronize with one another or aggregate partial results. The time for such an operation is a function of the group size, but not of the message length, as the latter is fixed. For example, in a barrier operation, a group of tasks synchronize with one another, by waiting until all tasks execute their respective barrier operation. In our performance measurements, we have divided the total communication time into two main parts. The first part is comprised of all point-to-point communications using MPI_ISEND, MPI_IRECV, and MPI_WAITALL primitives in the parallel code and is denoted by *PtoP_Comm*. These communications represent all the shared-boundary exchanges between nodes in the cluster that are needed to update such boundaries, as specified in the parallel algorithm. The second part groups both collective communication operations and aggregated computations. It is denoted by *Collective_Comm* and contains barrier and reduction operations, with the latter being implemented using the MPI_ALLREDUCE primitive. Collective_Comm represents the cost of synchronization between all nodes at the end of every simulation step and the calculation of important global output parameters using local values from each processor including volume coverage, tissue growth rate, average speed of multiple cell populations, and average number of cell collisions and aggregations.

We present in Fig. 11 the parallel execution time and the total communication time for $P = 20$ processors and $N \in \{150, 200, 250, 300, 330\}$. We also show in the same figure how the two components of communication time vary in this case. We observe the large increase in point-to-point communication time as the size of the cellular array is increased. This is due to the corresponding *quadratic* increase in the size of each shared boundary that needs to be exchanged with neighboring processors (every node in the cluster has two shared boundaries, each with a total of N^2 computational sites). In contrast, the collective communication time component only shows a small increase due mostly to system overhead since the related MPI routines are mainly unaffected with P being set equal to a constant value.

We display next, in Fig. 12, the parallel execution time and the total communication time for a $150 \times 150 \times 150$ cellular grid ($N = 150$) and $P \in \{2, 4, 8, 10, 20, 25, 50\}$. In addition, the two components of communication time are also shown in the same figure. We observe the large increase in collective communication time as the number of processors is increased. This is due to the corresponding increase in the time needed to communicate with more processors in order to coordinate and collect the results of the computations of many global output parameters as well as the synchronization necessary among a larger pool of nodes to yield up-to-date copies of all sub-domains at the end of each simulation step. In this situation, synchronizing all the nodes is required since the results of the next simulation step depend on the activities and processing occurring at the borders during the current simulation step. In contrast, we notice that the point-to-point communication component shows mainly small increases due to system related overhead. In essence, this is nearly a reversal of the behaviour observed in Fig. 11. However, more pronounced increases in point-to-point communication time are especially evident when there is a doubling

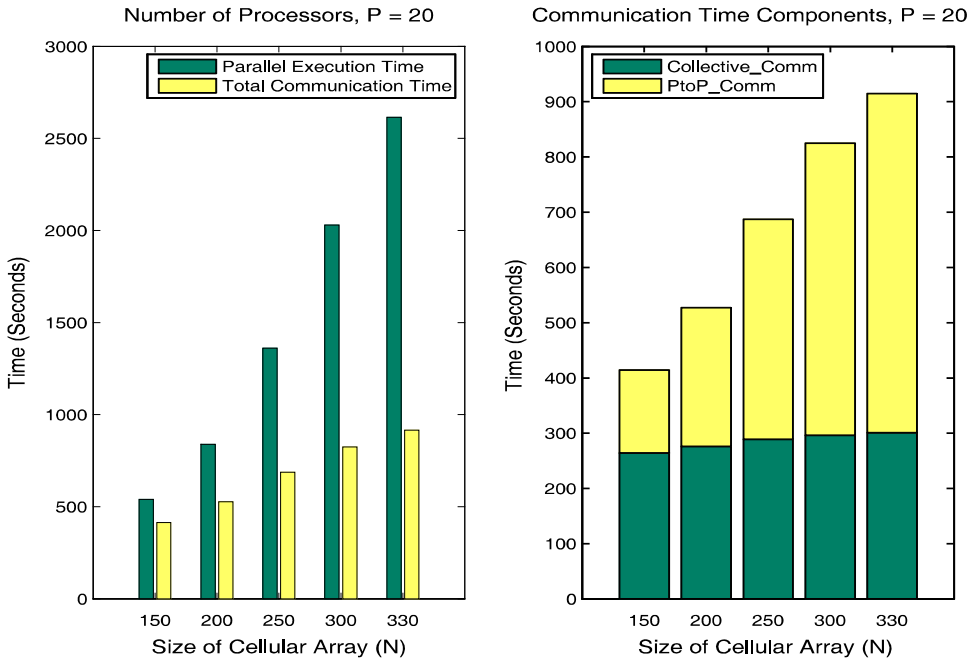

Fig. 11 (*Left*) Parallel execution time and total communication time versus cellular array size, N, for P = 20 processors. (*Right*) Components of communication time, *Collective_Comm* and *PtoP_Comm*, versus cellular array size, N, for $P = 20$ processors

of processor count (for instance, from $P = 25$ to 50). This can be explained by the fact that the increase in P might yield additional time delays in the completion of point-to-point communications even though the size of the messages are unchanged ($N = 150$).

8.2.3 Characterization of performance sweet spot

It is interesting to note that for the cellular array of size $150 \times 150 \times 150$, the speedup reaches its maximum value of approximately 3.94 when P is in the range from 10 to 20 processors and then starts to decrease for larger values of P due to the impact of *Amdahl's law* [55]. In this particular instance, and according to our performance data, we observe that the communication time dominates the parallel execution time and accounts for most of it (approximately 83 % when $P = 25$ and over 90 % when $P = 50$). This can be attributed to the large increase in the collective communication component. Consequently, the parallel execution time at $P = 25$ and $P = 50$ starts to increase and is greater than the parallel execution time at $P = 20$, as illustrated in Table 2 in bold type. We also observe the same outcome for the cellular array of size $200 \times 200 \times 200$, where the maximum speedup in this case is reached when $P = 25$ at an approximate value of 6.63.

Using Kuck's classification of performance levels and ranges, the minimum high-performance level of a parallel algorithm is defined by having a speedup equal to at least $\frac{P}{2}$ [56]. In addition, Amdahl's law may be broadly interpreted to mean the existence of a *point of diminishing returns* that represents a number of processors beyond which adding more processors to a computation produces so little performance gain that they are of questionable value [55]. This point can be used as some performance threshold to determine the appropriate processor count for a parallel machine configuration that would return fast performance results. To visualize this, imagine looking at the (S, P, N) performance space of a parallel algorithm and sweeping across the (P, N) plane while observing intervals of *good* performance. This process defines some region in the (P, N) plane that engenders a *sweet spot* of performance [56]. Therefore, based on the obtained speedup and efficiency results of our parallel algorithm, we could conjecture the following:

- For $150 \leq N \leq 330$, our parallel algorithm using the slab decomposition has a performance sweet spot defined by $P \in [2, 28)$.

The above statement may point to a weak *scalability* of the algorithm, especially in terms of the sizes of today's multicore and hybrid parallel systems. We believe that this might be explained by the choice of domain decomposition used and the well-known *surface effect* problem [49]. This effect generally deals with the surface-to-volume ratio of a domain decomposition technique. It is defined as the ratio of the total surface of a sub-domain to its total volume. A domain decomposition, with a high surface-to-volume ratio, would result in increased communication costs for its corresponding parallel algorithm. For the slab decomposition, we calculated its surface-to-volume ratio to be equal to $\frac{2P}{N}$. This value can be interpreted as *high* when compared to other regular decompositions that use a similar block-wise distribution of the cellular grid along two or all three dimensions. We believe

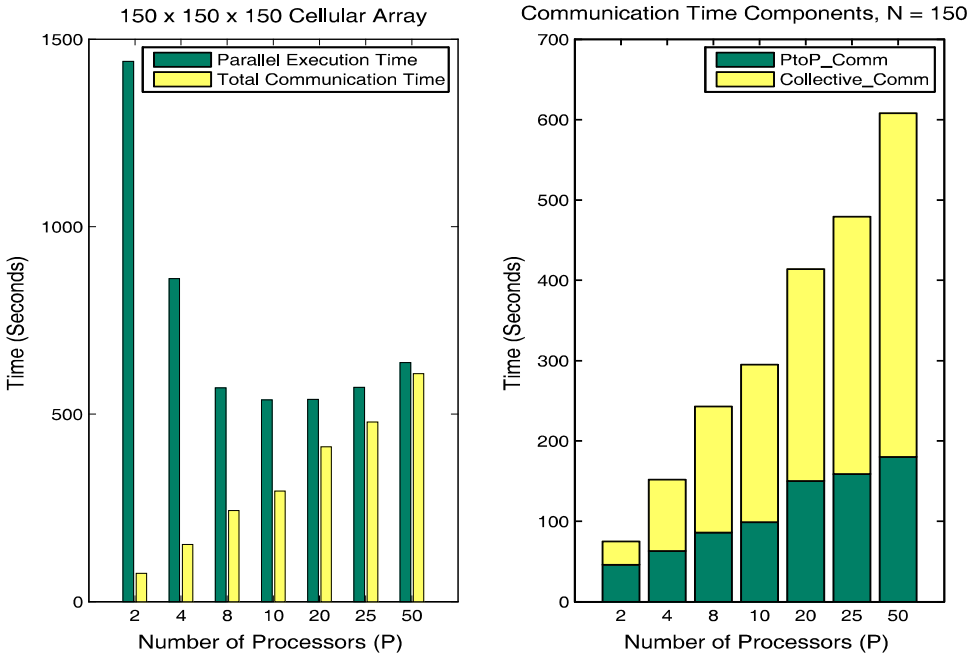

Fig. 12 (*Left*) Parallel execution time and total communication time versus number of processors, P, for a $150 \times 150 \times 150$ cellular array. (*Right*) Components of communication time, *Collective_Comm* and *PtoP_Comm*, versus number of processors, P, for a $150 \times 150 \times 150$ cellular array ($N = 150$)

that this is a key point that merits further study as part of our future work.

9 Conclusion and future work

We have presented in this article the parallelization of our three-dimensional computational model for the simulation of multicellular tissue growth, including a description of the main issues dealt with during this task as well as the computational techniques employed to generate deterministic simulation results. Performance results obtained on a cluster machine were *good* in terms of speedup and efficiency. The delivered performance can be mainly attributed to the fact that the parallel algorithm delays the exchange of cell movements and divisions across shared boundaries until all such cellular events are accounted for within a particular sub-domain. As a result of this strategy, the amount of communication between neighboring processors is reduced. Further, the exchanged messages were implemented using the non-blocking communication primitives of MPI (MPI_ISEND and MPI_IRECV), which allow for the overlapping of communication and computation. A feedback mechanism is used to handle any violation of the cellular automata rules, such as when more than one cell decides to move or divide into the same site.

To address the issue of scalability of the current parallel algorithm, we plan, as part of our future work, to implement various domain decomposition strategies and evaluate their performance using much larger cellular grids and processor counts. We will focus also on extending this model to include cell differentiation and cell death as well as its implementation on other parallel systems such as shared-memory and heterogeneous architectures, including multi-core CPU and GPU machines [57,58]. Moreover, we will work on integrating a visualization solution with the extended simulation model to assist researchers to explore the spatial and temporal domains of tissue growth in real time and to provide them with useful means to interpret and analyze simulation data and, potentially, to compare them with experimental results. This latter component of our research program is already underway and we have recently developed a visualization prototype for the base computational model using a single cell type [59,60].

Acknowledgments The author would like to gratefully acknowledge the continued support for this research work provided by the Research Center in the College of Computer & Information Sciences (under Project Number: RC121231) as well as the Deanship of Scientific Research, both at King Saud University. Further, we would like to thank the anonymous reviewers for their valuable comments on the manuscript as well as acknowledge the support of Simon Fraser University, Canada, for providing us with access to its HPC Cluster.

References

1. Hennessy, J.L., Patterson, D.A.: Computer Architecture: A Quantitative Approach, 5th edn. Morgan Kaufmann Publishers, San Francisco, CA (2012)
2. Brodtkorb, A.R., Dyken, C., Hagen, T.R., Hjelmervik, J.M., Storaasli, O.O.: State-of-the-art in heterogeneous computing. Sci. Prog. **18**(1), 1–33 (2010)
3. Wolfram, S.: Cellular Automata and Complexity: Collected Papers. Addison-Wesley, Reading, MA (1994)

4. Chaudhuri, P.P., Chowdhury, D.R., Nandi, S., Chattopadhyay, S.: Additive Cellular Automata: Theory and Applications, vol. 1. IEEE Computer Society Press, Los Alamitos, CA (1997)
5. Deutsch, A., Dormann, S.: Cellular Automaton Modeling of Biological Pattern Formation: Characterization, Applications, and Analysis. Springer-Verlag, Boston (2005)
6. Lysaght, M.J., Hazlehurst, A.L.: Tissue engineering: the end of the beginning. Tissue Eng. **10**(1–2), 309–320 (2004)
7. An, G., Mi, Q., Dutta-Moscato, J., Vodovotz, Y.: Agent-based models in translational systems biology. Wiley Interdiscip. Rev. **1**(2), 159–171 (2009)
8. Majno, G., Joris, I.: Cells, Tissues and Disease: Principles of General Pathology. Oxford University Press, Oxford (2004)
9. Page, E.H., Nance, R.E.: Parallel discrete event simulation: a modeling methodological perspective. In: Proceedings of the 1994 Workshop on Parallel and Distributed Simulation, pp. 88–93 (1994)
10. Hwang, M., Garbey, M., Berceli, S.A., Tran-Son-Tay, R.: Rule-based simulation of multi-cellular biological systems—a review of modeling techniques. Cell. Mol. Bioeng. **2**(3), 285–294 (2009)
11. Lauffenburger, D.A., Linderman, J.J.: Receptors: Models for Binding Trafficking and Signaling. Oxford University Press, New York (1993)
12. Levin, S.A., Grenfell, B., Hastings, A., Perelson, A.S.: Mathematical and computational challenges in population biology and ecosystems science. Science **275**(5298), 334–343 (1997)
13. Ben Youssef, B., Tang, L.: Simulation of multiple cell population dynamics using a 3-D cellular automata model for tissue growth. Int. J. Nat. Comput. Res. **1**(3), 1–18 (2010)
14. Tang, L., Ben Youssef, B.: A 3-D computational model for multicellular tissue growth. In: Proceedings of the 3rd International Symposium on Biomedical Simulation (ISBMS'06). Lecture Notes in Computer Science, vol. 4072, pp. 29–39 (2006)
15. Ben Youssef, B.: Simulation of cell population dynamics using 3-D cellular automata. In: Proceedings of the 6th International Conference on Cellular Automata for Research and Industry (ACRI'04). Lecture Notes in Computer Science, vol. 3305, pp. 562–571 (2004)
16. Frame, K.K., Hu, W.S.: A model for density-dependent growth of anchorage-dependent mammalian cells. Biotechnol. Bioeng. **32**, 1061–1066 (1988)
17. Cherry, R.S., Papoutsakis, E.T.: Modelling of contact-inhibited animal cell growth on flat surfaces and spheres. Biotechnol. Bioeng. **33**, 300–305 (1989)
18. Lim, J.H.F., Davies, G.A.: A stochastic model to simulate the growth of anchorage-dependent cells on flat surfaces. Biotechnol. Bioeng. **36**, 547–562 (1990)
19. Ruaan, R.C., Tsai, G.J., Tsao, G.T.: Monitoring and modeling density-dependent growth of anchorage-dependent cells. Biotechnol. Bioeng. **41**, 380–389 (1993)
20. Zygourakis, K., Bizios, R., Markenscoff, P.: Proliferation of anchorage-dependent contact-inhibited cells: I. Development of theoretical models based on cellular automata. Biotechnol. Bioeng. **38**(5), 459–470 (1991)
21. Hawboldt, K.A., Kalogerakis, N., Behie, L.A.: A cellular automaton model for microcarrier cultures. Biotechnol. Bioeng. **43**(1), 90–100 (1994)
22. Forestell, S.P., Milne, B.J., Behie, L.A.: A cellular automaton model for the growth of anchorage-dependent mammalian cells used in vaccine production. Chem. Eng. Sci. **47**(9–11), 2381–2386 (1992)
23. Lee, Y., Markenscoff, P., McIntire, L.V., Zygourakis, K.: Characterization of endothelial cell locomotion using a Markov chain model. Biochem. Cell Biol. **73**, 461–472 (1995)
24. Lee, Y., Kouvroukoglou, S., McIntire, L.V., Zygourakis, K.: A cellular automaton model for the proliferation of migrating contact-inhibited cells. Biophys. J. **69**(10), 1284–1298 (1995)
25. Chang, L., Gilbert, E.S., Eliashberg, N., Keasling, J.D.: A three-dimensional, stochastic simulation of biofilm growth and transport-related factors that affect structure. Microbiology **149**(10), 2859–2871 (2003)
26. Kansal, A.R., Torquato, S., Harsh IV, G.R., Chiocca, E.A., Deisboeck, T.S.: Simulated brain tumor growth dynamics using a three-dimensional cellular automaton. J. Theor. Biol. **203**(4), 367–382 (2000)
27. Cickovski, T.M., Huang, C., Chaturvedi, R., Glimm, T., Hentschel, H.G.E., Alber, M.S., Glazier, J.A., Newman, S.A., Izaguirre, J.A.: A framework for three-dimensional simulation of morphogenesis. IEEE/ACM T. Comput. Biol. Bioinformat. **2**(4), 273–288 (2005)
28. Motta, S., Pappalardo, F.: Mathematical modeling of biological systems. Brief. Bioinforma. **14**(4), 411–422 (2012)
29. Azuaje, F.: Computational discrete models of tissue growth and regeneration. Brief. Bioinforma. **12**(1), 64–77 (2011)
30. Drasdo, D., Kree, R., McCaskill, J.S.: Monte Carlo approach to tissue-cell populations. Phys. Rev. E **52**(6), 6635–6657 (1995)
31. Schaller, G., Meyer-Hermann, M.: Multicellular tumor spheroid in an off-lattice voronoi-DeLaunay cell model. Phys. Rev. E **71**(5 Pt 1), 051910 (2005)
32. Palsson, E.: A three-dimensional model of cell movement in multicellular systems. Future Gener. Comput. Syst. **17**, 835–852 (2001)
33. Beyer, T., Meyer-Hermann, M.: Delauny object dynamics for tissues involving highly motile cells. In: Chauviere, A., Preziosi, L., Verdier, C. (eds.) Cell Mechanics: From Single Scale-Based Models to Multiscale Modeling, pp. 417–442. CRC Press, Boca Raton, FL (2010)
34. Jiang, Y., Levine, H., Glazier, J.: Possible cooperation of differential adhesion and chemotaxis in mound formation of Dictyostelium. Biophys. J. **75**(6), 2615–2625 (1998)
35. Fu, Y.X., Chaplin, D.D.: Development maturation of secondary and lymphoid tissues. Annu. Rev. Immunol. **17**, 399–433 (1999)
36. Beyer, T., Schaller, G., Deutsch, A., Meyer-Hermann, M.: Parallel dynamic and kinetic regular triangulation in three dimensions. Comput. Phys. Commun. **172**(2), 86–108 (2005)
37. Drasdo, D., Jagiella, N., Ramis-Conde, I., Vignon-Clemental, I.E., Weens, W.: Modeling steps from benign tumor to invasive cancer: examples of intrinsically multiscale problems. In: Chauviere, A., Preziosi, L., Verdier, C. (eds.) Cell Mechanics: From Single Scale-Based Models to Multiscale Modeling, pp. 379–416. CRC Press, Boca Raton, FL (2010)
38. Marée, A.F., Hogeweg, P.: How amoeboids self-organize into a fruiting body: multicellular coordination in *Dictyostelium discoideum*. Proc. Natl. Acad. Sci. USA **98**(7), 3879–3883 (2001)
39. Tchuente, M.: Computation on automata networks. In: Fogelman-Soulie, F., Robert, Y., Tchuente, M. (eds.) Automata Networks in Computer Science: Theory and Applications, pp. 101–132. Princeton University Press, Princeton, NJ (1987)
40. Lee, Y., McIntire, L.V., Zygourakis, K.: Analysis of endothelial cell locomotion: differential effects of motility and contact inhibition. Biotechnol. Bioeng. **43**(7), 622–634 (1994)
41. Cheng, G., Ben Youssef, B., Markenscoff, P., Zygourakis, K.: Cell population dynamics modulate the rates of tissue growth processes. Biophys. J. **90**(3), 713–724 (2006)
42. Fox, G.C., Williams, R.D., Messina, P.C.: Parallel Computing Works!. Morgan Kaufmann Publishers, Inc, San Fransisco, CA (1994)
43. Quinn, M.J.: Parallel Programming in C with MPI and OpenMP. McGraw-Hill, Dubuque, IA (2004)
44. Pancake, C.M.: Is parallelism for you? IEEE Comput. Sci. Eng. **3**(2), 18–37 (1996)
45. van Hanxleden, R., Scott, L.R.: Load balancing on message passing architectures. J. Parallel Distrib. Comput. **13**(3), 312–324 (1991)

46. Chung, C.A., Lin, T.-H., Chen, S.-D., Huang, H.-I.: Hybrid cellular automaton modeling of nutrient modulated cell growth in tissue engineering constructs. J. Theor. Biol. **262**(2), 267–278 (2010)
47. Hoshino, T., Hiromoto, R., Sekiguchi, S., Majima, S.: Mapping schemes of the particle-in-cell method implemented on the PAX computer. Parallel Comput. **9**(1), 53–75 (1989)
48. van Hanxleden, R., Scott, L.R.: Correctness and determinism of parallel Monte Carlo processes. Parallel Comput. **18**(2), 121–132 (1992)
49. Fox, G.C., Johnson, M.A., Lyzenga, G.A., Otto, S.W., Salmon, J.K., Walker, D.W.: Solving Problems on Concurrent Processors: General Techniques and Regular Problems, vol. I. Prentice Hall, Englewood Cliffs, NJ (1988)
50. Knuth, D.E.: The Art of Computer Programming-Volume 2: Seminumerical Algorithms, 2nd edn. Addison-Wesley, Reading, MA (1981)
51. Fishmann, G.S., Moore, L.R.: An exhaustive analysis of multiplicative congruential random number generators with modulus 2^{31}-1. SIAM J. Sci. Stat. Comput. **7**(1), 24–45 (1986)
52. Ben Youssef, B., Sammouda, R.: Pseudorandom number generation in the context of a 3D simulation model for tissue growth. In: Proceedings of the 14th International Conference on Computational Science (ICCS 2014), Procedia-Computer Sciences, 29C, pp. 2391–2400. Elsevier, Edinburgh (2014)
53. Levesque, J.: High Performance Computing: Programming and Applications. Chapman & Hall, Boca Raton, FL (2011)
54. L'Ecuyer, P.: Random number generation. In: Gentle, J.E., Haerdle, W., Mori, Y. (eds.) Handbook of Computational Statistics, 2nd edn, pp. 35–71. Springer-Verlag, Berlin (2012)
55. Grama, A., Gupta, A., Karypis, G., Kumar, V.: Introduction to Parallel Computing, 2nd edn. Addison-Wesley, New York (2003)
56. Kuck, D.J.: High Performance Computing: Challenges for Future Systems. Oxford University Press, New York (1996)
57. Jin, H., Jespersen, D., Mehrotra, P., Biswas, R., Huang, L., Chapman, B.: High performance computing using MPI and OpenMP on multi-core parallel systems. Parallel Comput. 37(9), 562–575 (2011)
58. Dematté, L., Prandi, D.: GPU computing for systems biology. Brief. Bioinforma. **2**(3), 323–333 (2010)
59. Ben Youssef, B.: A visualization tool of 3-D time varying data for the simulation of tissue growth. Multimed. Tools Appl. 73(3), 1795–1817 (2014)
60. Ben Youssef, B.: Visualization of spatial patterns of cells using a 3-D simulation model for multicellular tissue growth. In: Proceedings of the 4th IEEE International Conference on Multimedia Computing and Systems (ICMCS'14), pp. 367–374. IEEE Xplore (2014)

Belgacem Ben Youssef is an Associate Professor in the Department of Computer Engineering, College of Computer & Information Sciences at King Saud University, Riyadh, Saudi Arabia. He received his PhD from the Department of Electrical & Computer Engineering, Cullen College of Engineering, University of Houston, Houston, Texas, USA. He was previously an Assistant Professor in both the School of Interactive Arts & Technology and the TechOne Program at Simon Fraser University, Vancouver, British Columbia, Canada. His research interests include parallel/multicore computing, computational tissue engineering, visualization, digital signal processing, and spatial thinking in learning and design. He has two years of industrial experience in software development and technical project management. He is a member of the IEEE, IEEE Computer Society, and the ACM.