

Efficient approaches to simulating individual-based cell population models



Daniel Harvey
Mansfield College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy
Trinity 2013

Abstract

‘Efficient approaches to simulating individual-based cell population models’, Daniel. G. Harvey, submitted for the degree of Doctor of Philosophy. Trinity 2013.

Computational modelling of populations of cells has been applied to further understanding in a range of biological fields, from cell sorting to tumour development. The ability to analyse the emergent population-level effects of variation at the cellular and subcellular level makes it a powerful approach. As more detailed models have been proposed, the demand for computational power has increased. While developments in microchip technology continue to increase the power of individual compute units available to the research community, the use of parallel computing offers an immediate increase in available computing power.

To make full use of parallel computing technology it is necessary to develop specialised algorithms. To that end, this thesis is concerned with the development, implementation and application of a novel parallel algorithm for the simulation of an off-lattice individual-based model of a population of cells.

We first use the Message Passing Interface to develop a parallel algorithm for the overlapping spheres model which we implement in the Chaste software library. We draw on approaches for parallelising molecular dynamics simulations to develop a spatial decomposition approach to dividing data between processors. By using functions designed for saving and loading the state of simulations, our implementation allows for the parallel simulation of all subcellular models implemented in Chaste, as well as cell-cell interactions that depend on any of the cell state variables. Our implementation allows for faithful replication of model cells that migrate between processors during a simulation. We validate our parallel implementation by comparing results with the extensively tested serial implementation in Chaste.

While the use of the Message Passing Interface means that our algorithm may be used on shared- and distributed-memory systems, we find that parallel performance is limited due to high communication costs. To address this we apply a series of optimisations that improve the scaling of our algorithm both in terms of compute time and memory consumption for given benchmark problems.

To demonstrate an example application of our work to a biological problem, we extend our algorithm to enable parallel simulation of the Subcellular Element Model (S.A. Sandersius and T.J. Newman. *Phys. Biol.*, 5:015002, 2008). By considering subcellular biomechanical heterogeneity we study the impact of a stiffer nuclear region within cells on the initiation of buckling of a compressed epithelial layer. The optimised parallel algorithm decreases computation time for a single simulation in this study by an order of magnitude, reducing computation time from over a week to a single day.

Contents

1	Introduction	7
1.1	Overview of thesis	10
1.2	Structure of thesis	13
2	Individual-based models of populations of cells	14
2.1	Lattice-based models	15
2.1.1	Cellular automata	15
2.1.2	The cellular Potts model	17
2.2	Off-lattice models	20
2.2.1	Cell centre models	21
2.2.2	Vertex dynamics models	31
2.2.3	Agent-based models	33
2.2.4	Subcellular element models	35
2.3	Continuum approximation of discrete models	38
2.4	Summary	40
3	Overlapping spheres model of a cell population	42
3.1	Model details	42
3.2	Numerical solution of model equations	45
3.3	Software implementation	46
3.3.1	Chaste	46
3.3.2	Implementation of cell-based models	50

3.3.3	Physical units	54
3.3.4	Simulations in Chaste	55
4	Parallelising an off-lattice model of a cell population	60
4.1	Introduction	60
4.2	Previous work	63
4.2.1	Parallel approaches in molecular dynamics	63
4.2.2	Parallel implementation of individual-based models	74
4.3	Distributed algorithm	79
4.3.1	Data decomposition	79
4.3.2	Halo data and cell migration	82
4.3.3	Data communication	83
4.3.4	Cell identifiers	85
4.3.5	Output	86
4.3.6	Simulation in fewer spatial dimensions	87
4.4	Results	87
4.4.1	Validation	87
4.4.2	Simulation of a simplified cell population model	91
4.4.3	Implementation of the full model	93
4.5	Discussion	101
5	Optimising the distributed algorithm	104
5.1	Introduction	104
5.2	Efficient construction	105
5.3	Efficient containers	109
5.4	Initial scaling results	112
5.5	Binary archiving	119
5.6	Non-blocking communication I	122
5.7	Non-blocking communication II	130

5.8	Load-balancing	133
5.9	Discussion	139
6	Case study: the instigation of colorectal crypt fission	145
6.1	Introduction	145
6.1.1	Crypt structure and fission	146
6.1.2	Previous modelling work	148
6.1.3	Aim of work	156
6.2	Materials and methods	159
6.2.1	The subcellular element model	160
6.2.2	Model development	162
6.2.3	Implementing the ScEM in Chaste	165
6.3	Results	167
6.3.1	Effect of the cell nucleus	168
6.3.2	Effect of relative cell size	172
6.3.3	Effect of relative cell stiffness	176
6.3.4	Computational tractability	176
6.4	Discussion	180
7	Discussion	183
7.1	Parallelisation	184
7.2	Application case study	186
A	Scaling the parameters of the subcellular element model	189
A.1	Scaling the equilibrium distance r_{eq}	189
A.2	Scaling the elastic potential	190
A.3	2D creep stress experiments	191

Acknowledgements

Whilst I maintain that the research presented in this thesis is my own work, I cannot take all the blame: there are a number of people without whom it would not have been completed.

Firstly, I would like to thank my three supervisors: James, Joe and Alex. James, your endless enthusiasm, ideas and D.I.Y. stories kept me going. Joe, your expert scientific computing knowledge made the technical aspects of the thesis possible (in particular Chapter 11 of Pitt-Francis and Whiteley [2012]). Alex, your encyclopedic mathematical and computational biology knowledge, as well as your grammatical punctiliousness, raised the standard of this thesis immeasurably.

Secondly, I would like to thank the EPSRC for the funding without which I would not have been able to embark on this project. I hope it was money well spent.

Thirdly, I would like to thank the rowers who made my time at Oxford so much fun. The lightweight Blue Boats of 2009 and 2010, Isis 2011, and the Blue Boat 2012, and the numerous crews of Mansfield College Boat Club.

Lastly, I would like to say thank you those people who supported me throughout the last four years. To Lucy, for putting up with me rowing full-time and completing a degree without once complaining that I only ever came home to sleep. To Gary, my pastoral supervisor for dragging me to the pub for Early Beer Friends Club. To Dad, for your support and enthusiasm. Finally, to Mum, for being there when completing this thesis felt like an impossible task.

Declaration

This thesis is submitted to the Department of Computer Science, University of Oxford, for the degree of Doctor of Philosophy. This thesis is entirely my own work, and, except where otherwise indicated, describes my own research.

The research in this project has principally been carried out using the Chaste software library, which is a collaborative project involving members of the Computational Biology Group and the Wolfson Centre for Mathematical Biology at the University of Oxford. During my research I have been involved in all aspects of the development of the library, from planning, developing and maintaining the code to assisting in the organisation of a workshop. Furthermore, I have assisted in the development of functionality for the recently published software article in *PLoS Comput. Biol.* on Chaste [Mirams et al., 2013]. The work described in this thesis, and particularly the novel research in Chapters 4, 5 and 6, has been carried out independently of these collaborative efforts.

The code developed as part of the work on parallelising a model of a cell population is available with the Chaste source code at <https://chaste.cs.ox.ac.uk/trac/browser/trunk> from which it is possible to navigate to the appropriate classes. The key classes that have formed part of the development in this part of my research are:

- `global/src/ObjectCommunicator.hpp`: a class for communicating C++ objects.
- `mesh/src/utilities/DistributedBoxCollection.hpp`: a class for decomposing the spatial domain and calculating cell neighbourhood topology.
- `mesh/src/common/NodesOnlyMesh.hpp`: a class containing the location associated with each cell in a spatially distributed manner.
- `cell_based/src/population/NodeBasedCellPopulation.hpp`: a class containing and updating the set of cells, and managing inter-processor communication.

The code developed for the case study in Chapter 6 is publically available to download as a ‘bolt-on’ project from <https://chaste.cs.ox.ac.uk/trac/browser/projects/Sem>

Glossary

AD Atom Decomposition. The method of parallelising a molecular dynamics simulation by assigning to each processor a subset of the atoms or molecules to simulate. 67, 69–73, 77, 133, 139

Bandwidth The rate at which a communication network can transfer data, typically measured in bits/s. 78

Computation time The time taken to execute a program, or some part of a program. See *wall time*. 62, 95

CPM Cellular Potts Model. A lattice-based model of cell populations developed by Graner and Glazier [1992]. 9, 17–20, 23, 40, 41, 75

CPU Central Processing Unit. The main processing unit in a computer, which carries out the basic computational tasks. A CPU can contain more than one processor. 8, 63, 73, 74, 77, 116

CRC Colorectal Cancer. Uncontrolled cell growth in the colon or rectum. 145, 146, 155

FD Force Decomposition. The method of parallelising a molecular dynamics simulation by assigning to each processor a subset of the possible atom or molecule interactions to compute. 67, 70–73, 133, 139

GPGPU General Purpose Graphics Processing Unit. A graphics card that has been adapted for use in general computing tasks other than graphics rendering. It is

-
- used to complement the power of the central processing unit. 35, 37, 67, 73, 74, 77, 78, 142
- GUI** Graphical User Interface. A graphics based computer program for controlling and interacting with software, as opposed to code- or text-based interaction. 19, 30
- HPC** High-Performance Computing. Computing at the leading edge of current processing capacity. Such computers typically contain a large number of processors with either a large shared-memory, or a distributed-memory store. 35, 61, 62, 66, 67, 73–75, 77, 78, 114, 120, 141, 169
- Latency** The time delay in a network between a communication being sent and being received. 78
- MCS** Monte-Carlo Step. In a cellular Potts model, a Monte-Carlo Step is equivalent to N trial spin changes, where N is the number of lattice sites. 18, 20
- MD** Molecular Dynamics. Computer simulations of the movement of atoms and molecules by numerically solving an equation of motion for each atom or molecule. 10, 62–65, 67, 69–71, 73, 74, 77, 78, 134
- MPI** (The) Message Passing Interface. A model for communication in parallel computer architectures [Gropp et al., 1999]. 77, 83, 84
- ODE** Ordinary Differential Equation. 19, 23, 45, 55
- PDE** Partial Differential Equation. 16, 18, 19, 27, 30, 32, 39, 46, 53
- Relative speed-up** The speed-up achieved by using multiple processors relative to the problem size considered. For a problem of size N_p on p processors that takes a time t_p to execute, relative speed-up is given by $t_1 N_p / t_p N_1$. 113

RSS Resident Set Size. The amount of memory currently being used by a computational process. 106, 108

ScEM Subcellular Element Model. A model of populations of cells that represents each cell as the union of a number of subcellular elements. Developed by Newman [2005], the model is particularly adept at representing the response of cells to mechanical stimuli over a range of time scales. 12, 36, 37, 41, 63, 77, 158–163, 165, 169, 180, 187, 191

SD Space Decomposition. The method of parallelising a molecular dynamics simulation by assigning to each processor a subset of the physical domain, and any atom or molecule that lies in that domain. 67, 70–73, 76, 78, 79, 81, 101

Speed-up A measure of the performance improvement of a program obtained by increasing the total number of compute processors used. Defined as the ratio t_1/t_p where t_1 is the computation time for a program in serial, and t_p is its computation time on p processors. 62

STL Standard Template Library. A library of containers, algorithms, functionals and iterators implemented in C++. 109

TDD Test Driven Development. An approach to software development that enforces the writing of unit tests before source code to ensure correct functionality [Martin et al., 2012]. 47

Wall time The time elapsed between the beginning and end of a computation operation. As opposed to, e.g. CPU time as the sum of computation time across all processors. 95

Conventions

For clarity, throughout this thesis we shall refer to a ‘processor’ to mean both a physical processor and a computational ‘process’ on a single physical processor. This distinction is necessary for modern computers which allow multi-threading of multiple processes on a single processor. However, throughout all chapters, unless otherwise stated, we will always construct a parallel simulation with a single process operating on each physical processor and so any confusion should be avoided.

Text written in true type font, such as `AbstractCellPopulation`, refers to the name of a class in Chaste. Methods in the software are appended with brackets, such as `AddForceContribution()`, and member variables of a class are prepended with an `m`, such as `mCells`.

Chapter 1

Introduction

Biological function in multicellular organisms arises through the actions of individual cells and their interaction with each other and their environment. While our understanding of many of these properties has been advanced through *in vivo* and *in vitro* experimentation, there is a growing role for mathematical and computational modelling in explaining experimental results and generating hypotheses that motivate further experimental studies. In particular, by modelling at the level of the cell, individual-based approaches enable studies of phenomena that rely on heterogeneity within the population. For example, some of the early work in this field computationally validated the differential adhesion hypothesis of Steinberg [1964] which explains the self-sorting of mixed aggregates of cells of different types. The sorting can be explained by an individual-based model where cells of distinct types experience different adhesion strengths with each other [Sulsky et al., 1984, Glazier and Graner, 1993, Goel et al., 1970].

In step with advances in computational power and available experimental data, more detailed individual-based models have been proposed that still focus at the single cell level but are extended to include representation of subcellular processes, such as the growth and division of single cells, and microenvironmental conditions at the tissue scale such as nutrient and morphogen concentrations. Detailed multi-scale models of healthy and pathological growth of cell populations may be created by coupling these approaches

together. For example, the growth patterns of *in vitro* models of early tumour development can be explained using an individual-based model where the growth and division of individual cells, and hence the growth of the population as a whole, is regulated by both mechanical cell-cell interactions and the limited availability of diffusing oxygen and glucose in the microenvironment [Höehme and Drasdo, 2010b].

Individual-based models may be broadly split into two classes: lattice- and off-lattice-based. The former imposes that the physical representation of cells lies on a fixed lattice, with a single lattice site representing anything from a mesoscopic section of a single cell, to multiple single cells. The latter approach allows each cell to move continuously in space by representing the cell either as a coordinate for its centre or as a collection of coordinates. The off-lattice approach allows a more natural and realistic representation of mechanical cell-cell interactions, such as adhesion between cells and the resistance of cells to compression by their neighbours. However, this comes at a computational cost. In addition to simulating the models of each single cell, it is also necessary to update the cell neighbourhood topology. That is, which cells are physically in contact with each other, or are able to interact with each other through local signalling. More detailed force laws between neighbouring cells further adds to computational cost.

While the computational power of single Central Processing Units (CPUs) continues to grow and increase the speed and scale of possible simulations, parallel computing offers increases in available processing capacity by allowing multiple individual processors to work in unison. This provides a simple solution to running large numbers of independent simulations of cell population models by splitting the work between different units. However, to tackle single simulations which are computationally intractable requires the coordination of many processors working together. This approach has had significant success in increasing both the speed of single simulations and the scale of problem that may be tackled in other related fields, such as Molecular Dynamics (MD) [Hess et al., 2008]. However, the same success has not been replicated in providing a tool for parallel simulation of cell population models. Recent developments have included parallel schemes

for the lattice-based Cellular Potts Model (CPM) in the open-source CompuCell3D software package [Chen et al., 2007], and in the FLAME modelling environment which is able to generate parallel simulation code for a general class of agent-based model [Coakley et al., 2012]. Parallel schemes for a single specific model have been able to utilise the power of multiple processing units, but do not extend beyond their use for a specific application [Höehme and Drasdo, 2010a, Christley et al., 2010].

The Chaste software library has been developed by members of the Computational Biology Group and the Wolfson Centre for Mathematical Biology at the University of Oxford, and provides an open-source, maintained and extensible suite of C++ classes that can be used for simulating individual-based cell population models, as well as providing functionality for modelling cardiac electrophysiology [Mirams et al., 2013]. Although the models in these two distinct areas are quite different, their simulation requires some common functionality which is provided by a range of ‘core’ libraries in Chaste. A growing range of lattice-based and off-lattice models of cell populations have been implemented, allowing a user to choose the most appropriate model for a given problem, and even to compare results by switching between different models at the cell level, whilst maintaining the same model at the subcellular level [Osborne et al., 2010].

In terms of performance, Chaste benefits from being written in C++, as opposed to an interpreted language. When a simulation is compiled, a number of optimisations can be applied which lead to significant performance gains. Nevertheless, parallel computing offers the opportunity to increase the speed with which a model may be simulated, and the scale of model that may be considered. This has recently been demonstrated in the cardiac electrophysiology element of the library [Bernabeu et al., In Press 2013]. The range of available subcellular and cell-cell interaction models in Chaste, its extensibility, and the fact that it is maintained and open-source make Chaste the ideal library in which to develop a parallel algorithm for an individual-based cell population model that will be of use beyond a single project.

1.1 Overview of thesis

In this thesis we parallelise an individual-based model of cell populations that treats each single cell as a point in space, which represents the centre of its nucleus. The cell volume is modelled as a sphere around this central point, and cells interact mechanically with each other through force laws which give rise to a Langevin equation of motion for each cell. Our parallel scheme assumes only that these cell to cell interactions are short-ranged in nature, i.e. cells separated by more than some fixed distance experience no direct interaction with each other. This type of model has been used in a number of computational studies, such as modelling growing tumours [Galle et al., 2005, Höehme and Drasdo, 2010b] and organisation in the colorectal crypts [Buske et al., 2011].

Enabling a single simulation of this model to execute in parallel presents two major challenges: how to split work between the different processors, and how to allow the processors to communicate to ensure the simulation is carried out correctly and efficient use is made of each processor. These problems can either be handled by splitting the data between processors and having them execute the same commands on all data, or alternatively splitting the tasks that constitute the simulation between the processors and managing the flow of data between processors.

Although there are few examples of parallelised off-lattice cell population models in the literature, there are a number of analogies between these models and those used in Molecular Dynamics (MD) where a mature field of parallel computation exists. In particular, parallel MD schemes deal with free-moving points in space driven either by an interaction force or an interaction potential between the points. We review techniques used in this field which we use to inform our development of a parallel scheme by dividing the physical domain space between different processors and assigning to each processor those cells located in some region of space. This approach benefits from minimal communication between processors, because only cells near the physical space boundary between processors must be communicated. Furthermore, this approach can benefit from efficient memory use because there is no requirement to replicate large amounts of cell data across

all processors.

However, this approach presents a number of difficulties, which we also address in this thesis. Firstly, as we have noted, the decomposition of data will mean that some pairs of cells that interact with each other are stored on separate processors, and furthermore, cells may move between space owned by one processor to space owned by another. Since the Chaste library allows for general short-range interactions between cells we must enable sharing of individual cell data between separate processors through communication. This is complicated by the general purpose class structure of Chaste, where different models of, for example, the cell cycle, may be used in each cell. We cannot enumerate these different model types as the library is under constant development, with new subcellular models regularly being added. We therefore take a flexible and extensible approach to communication by using methods written into every class in Chaste to save the state of an object to file, called serialisation. This allows us to create strings of data that completely represent the cell objects, and can be sent and received between processors using message passing.

This approach gives rise to relatively high communication costs, which limit the speed-up in simulation execution time that may be gained by using parallel computing. We address this by investigating a number of optimisations to the implementation of our algorithm, including restricting our algorithm to homogeneous parallel computers to speed the creation and decoding of messages, and by overlapping communication and computation functions to minimise the cost of communication.

The second problem we address is that of load-balancing: the decomposition of cell data based upon the cell's location does not guarantee an even distribution of work between processors. In particular, for growing populations of cells that exhibit large-scale movement of cells we see that a static decomposition of space leads to large imbalances in load between processors, meaning full use is not made of all available processing power consequently limiting speed-up. We take a simple initial approach to dynamically balance the load between processors by adjusting the decomposition of space to attempt to

equalise the number of cells located on each processor.

Having developed, implemented and optimised our parallel algorithm we carry out a biological modelling case study which demonstrates its application and its flexibility. As a result of collaboration with experimentalists, we investigate the role of different mechanical properties of subcellular regions on a layer of epithelial cells under compression. Specifically, we are interested in an event that occurs in the epithelial lining of the intestine, where small invaginations called crypts are seen to buckle and undergo fission into two or more new crypts. This fissioning behaviour is observed to be particularly prevalent in developing adenomas, a precursor to colorectal cancer, and is understood to be the process through which an aberrant mutation in a single cell spreads from one crypt to multiple crypts [Greaves et al., 2006].

Recent developments in *in vivo* tissue cultures allow this fission process to be observed directly [Sato et al., 2009], and in particular it has been noticed that these ‘buckling’ events occur as cells become compressed down to the width of their nucleus, as a result of growth and division within the layer. Based upon the observation that cell nuclei are mechanically stiffer than the surrounding cytoplasm [Maniotis et al., 1997], we investigate whether a stiffer region within cells can cause an out-of-plane deflection that may trigger buckling and fission events.

In order to test our hypotheses, we use the Subcellular Element Model (ScEM), a more detailed individual-based model that represents cells as a collection of mesoscopic elements, rather than a single point in space [Newman, 2005]. This model more accurately captures the passive biomechanical properties of individual cells within a population [Sandersius and Newman, 2008]. Furthermore, the model allows us to incorporate mechanical heterogeneity within individual cells to model a stiffer nuclear region. The detail of the model means that a single simulation in Chaste would take around a week to complete using the original serial algorithm, precluding an extensive study. Extending our parallel algorithm for the cell centre model to the ScEM is straightforward due to the modular nature of Chaste, and along with other optimisations presented in this thesis,

allows us to reduce the time taken for a single simulation to under a day.

1.2 Structure of thesis

The remainder of the thesis is structured as follows. In Chapter 2 we review the field of individual-based modelling of cell populations and consider the range of models available, their applicability and where appropriate how they have been made available to the wider scientific community in the form of software. In Chapter 3 we specify the model that we consider for parallelisation, how the model is simulated and introduce the structure and development strategy of the Chaste software library. Chapter 4 comprises the first novel research chapter, and describes the development of a distributed parallel algorithm for simulating the model, and concludes with a survey of its performance for a range of benchmark simulations as the number of processors is increased. In Chapter 5 we specify two benchmark problems and carry out a number of optimisations of our parallel algorithm and its implementation that improve the performance by a factor of approximately 100 on 32 processor cores, over the original serial algorithm. We also develop a simple load-balancing algorithm that allows our parallel algorithm to improve performance of simulations of growing or irregularly shaped spatial distributions of cells. In Chapter 6 we carry out a modelling case study which uses our parallel algorithm to study the effect of stiffer cell nuclei on a layer of cells under compression, using a refinement of the model in Chapter 3 which allows for a more detailed representation of cellular mechanics, and the introduction of subcellular mechanical heterogeneity. Finally, in Chapter 7 we discuss the conclusions of our work and ideas for future development of both the parallel algorithm and the modelling case study.

Chapter 2

Individual-based models of populations of cells

The primary goal of this thesis is to develop an efficient parallel algorithm for simulating a class of off-lattice cell population models within the Chaste framework. The type of model we consider is a single example from a wider field of individual-based models where the fundamental unit is the cell. In this chapter we aim to review these models, and, where appropriate, how they have been made accessible to the wider scientific community through open-access software tools and libraries. We consider how each model has been developed and how they have been extended, which typically includes coupling to models at the subcellular and tissue scales. Although not the focus of this review, we describe and refer to these models at different scales as appropriate. By focusing on the constitutive assumptions and model implementations, rather than the biological insights gained through their use, we have a broad scope of application areas, but they tend to be focused on epithelial development and the initiation of carcinomas (epithelial cancers).

In this chapter we do not consider the continuum class of models. These models are concerned with the average behaviour of populations of cells, and are typically formulated as differential equations describing how the density of cells changes in space and time. In some cases, by applying analytical techniques to such models, it is possible to completely

characterise the behaviour of the model for different parameter ranges – something which is not possible for most individual-based models. Furthermore, in the case where an analytic treatment is not possible, numerical solution of continuum models is usually less computationally costly than individual-based models as they tend to have many fewer degrees of freedom. However, by treating the population as a continuum, such models cannot capture the emergent tissue-level behaviour as a result of differences between individual cells in a population. For a detailed review of continuum models of tissue growth and dynamics we refer the reader to reviews by Jones and Chapman [2012] or, in the context of cancer modelling, the review by Roose et al. [2007].

We split our review initially into two broad classes of individual-based models: those that restrict cells to lie on a (typically square or hexagonal) fixed lattice (lattice-based models) and those that are formulated so that cells may move continuously throughout the model domain (off-lattice models).

2.1 Lattice-based models

2.1.1 Cellular automata

The simplest individual-based models used in the context of modelling tissues and populations of cells are cellular automata. Initially proposed by von Neumann [1966], these models are explicitly discrete in time, space and the state of the agents: the state of each lattice site is updated using probabilistic rules that incorporate the state of all other lattice sites in the model, although typically only near-neighbours affect the update of each site. Updating all lattice sites in parallel defines a discrete step forward in time. Applied to cell populations, the state of each lattice site typically denotes the type of cell that occupies that site, and transition rules may implement, for example, differential adhesion between cells [Rogers and Goel, 1978], or population growth through increasing cell numbers [Düchting and Vogelsaenger, 1985]. Further extensions consider cells as having their own states which they carry as they move through the lattice, representing, for example,

their progress through the cell cycle [Düchting and Vogelsaenger, 1985].

While such models represent the first steps in individual-based computational modelling of cell populations, they are by no means obsolete. For example, Alarcón et al. [2003] developed a model for tumour growth that includes blood flow through a vascular network and showed how tumour growth and invasion are limited by spatial variation in the availability of nutrients as a result of a vascular network. Although the transport of oxygen at the tissue-level is modelled by a quasi-steady-state Partial Differential Equation (PDE), the cell-level is modelled by a cellular automaton, with state update rules for the cell cycle coupled to the local oxygen concentration. A similar hybrid cellular automaton model developed by Smallbone et al. [2007] reproduces the cycle of transitions of cancerous cells between hypoxia, glycolysis and acidosis in early carcinomas. By including a detailed model of cellular metabolism based on the local concentration of oxygen and glucose diffusing from the vasculature, as well as including the effect on local pH of anaerobic respiration of cancerous cells, the authors are able to study the prevalence of hypoxia in early tumour development.

The simplicity of cellular automata makes implementing algorithms to simulate them a relatively low cost process. This most likely explains the lack of published, extensible software for these models: the cost of learning and adapting existing software for one's own needs is comparable to writing it from scratch. Software tools such as Cellab [Rucker and Walker] are no longer supported for modern operating systems, suggesting a lack of demand. Recently, Figueredo et al. [2013] reported an implementation of a multi-scale model based on a cellular automata in the Chaste software library which, to our knowledge, is the only such maintained, tested, open-source implementation of a cellular automata for modelling populations of cells.

The principal benefits of cellular automata are their simplicity and the computational speed that results from a relatively small state space compared to their off-lattice counterparts. This simplicity made them tractable when computing power was extremely limited, and now allows fast computation of highly detailed, multi-scale models, such as

those of Alarcón et al. [2003] and Smallbone et al. [2007]. Furthermore, as observed by Ermentrout and Edelstein-Keshet [1993], their formulation in terms of rules rather than equations means that cellular automata provide a bridge between verbal, non-rigorous models and rigorous mathematical models and hence provide a tool for research that helps to bridge the gap between medical clinicians and physical scientists.

Nevertheless, cellular automata constitute a highly simplified abstraction of reality, rendering them unsuitable for modelling certain phenomena. The most striking deficiency is their simplification of cell movement, shape and volume. In a cellular automaton, a cell's shape is fixed and it has constant volume – there is no incorporation of the effect of the local environment on these properties. Furthermore, cell motion is restricted to large, discrete steps. Therefore, the motion of a single cell causes long-distance effects on other cells as space must be made to accommodate the movement.

2.1.2 The cellular Potts model

The limitations highlighted above are partially overcome by the CPM [Graner and Glazier, 1992]. Originally developed for modelling processes in solid-state physics, the CPM was extended by Graner and Glazier [1992] to simulate cell sorting based on the differential strength of adhesion between two cell types. Rather than a cell occupying a single lattice site, as with cellular automata, a cell is defined to be the union of all lattice sites (i, j) with a particular ‘spin’ $\sigma(i, j)$. In addition to the spin, a lattice site is also labelled with the cell type $\tau(\sigma(i, j))$ of the cell occupying that site. Each cell type is assigned a target area and an adhesion strength with neighbouring sites of the same and different cell types.

Rather than using explicit rules, the evolution of the CPM is governed by a stochastic energy minimisation process. A Hamiltonian $\mathcal{H}$ is defined which assigns an energetic cost to unfavourable configurations such as deviation of a cell from its target area, or adhesion with different cell types. For example, Graner and Glazier [1992] use the following Hamiltonian for cell sorting experiments:

$$\mathcal{H} = \sum_{\text{neighbours}} J(\tau(\sigma(i, j)), \tau(\sigma(i', j'))) [1 - \delta_{\sigma(i, j), \sigma(i', j')}] + \lambda \sum_{\sigma} [a(\sigma) - A_{\tau(\sigma)}]^2 \theta(A_{\tau(\sigma)}),$$

where $\sigma(i, j)$ is the spin associated with lattice site (i, j) , J represents surface adhesion between different cell types τ , λ is an elastic parameter, $a(\sigma)$ is the area of cell σ and $A_{\tau(\sigma)}$ is the target area for that cell type. θ is the Heaviside function. The first term is the energy associated with cell-cell adhesion, while the second characterises the resistance of cells to deformation.

A Metropolis-Hastings algorithm [Metropolis et al., 1953] is used to attempt to minimise $\mathcal{H}$: an iteration of the algorithm consists of a lattice site and a neighbour being selected at random and the change in total energy resulting from copying the spin of the first site to the second, $\Delta\mathcal{H}$, calculated. The spin change is accepted with probability $\min(1, e^{-\Delta\mathcal{H}/\kappa T})$, where T is a thermal parameter characterising fluctuations in the system and κ is the Boltzmann constant. The temperature T may be varied to fully explore the state space and find the global minimum of the system. The simulation progresses in units of Monte-Carlo Step (MCS), with one MCS being equivalent to N trial spin changes, where N is the number of lattice sites.

Using the CPM, Graner and Glazier [1992] were able to computationally reproduce the sorting of mixed aggregates of cells based on the differential adhesion hypothesis proposed by Steinberg [1964]. As with cellular automata, the original model has been vastly extended to include cellular mechanisms such as growth, division, death, haptotaxis and chemotaxis by including further energetic constraints within the Hamiltonian [Izaguirre et al., 2004]. The CPM has also been coupled to subcellular models of gene regulation networks and reaction-diffusion PDE models of the transport of nutrients and morphogens, thus allowing non-linear feedback between multiple spatial scales. In particular this has enabled the integration of many isolated models at different spatial and temporal scales into a single simulation of complex developmental processes, such as somite

formation [Hester et al., 2011].

One of the reasons for the popularity of the CPM is its availability in the comprehensive CompuCell3D software package [Izaguirre et al., 2004, Swat et al., 2012]. In addition to the basic CPM, the software allows: the definition of multiple cell types with associated parameters; the coupling of a reaction-diffusion system that is numerically solved on either a finite-difference or finite element grid; and the inclusion of Ordinary Differential Equation (ODE) or discrete-state subcellular models, including models developed using the Systems Biology Workbench [Sauro et al., 2003]. Furthermore, the software has a user-friendly Graphical User Interface (GUI) and multiple visualisation options enabling model set-up and analysis. For computational speed, the model has been parallelised using a distributed algorithm, with good efficiency on tens of processors [Chen et al., 2007]. We provide more details on the parallel scheme in Section 4.2.2. Recently, de Back et al. [2012] have developed a similar software environment, Morpheus, which integrates subcellular ODE models into a CPM that can be simulated on 2D square or hexagonal lattices and in 3D on cubic lattices. Tissue-level diffusion of nutrients or metabolites can be included using reaction-diffusion PDEs. Similarly to CompuCell3D, it is possible to ‘plug-in’ addition model variations, and to define models using a mark-up language, rather than programming directly to ease the use of the software for non-experts.

The CPM shares many of the benefits of cellular automata in terms of both computational and conceptual simplicity, but also reduces the lattice artifacts seen in cellular automata. Computationally, a fixed lattice enables fast simulation compared to off-lattice models, although assigning multiple sites to a single cell reduces speed compared to an automaton for a fixed number of cells. Conceptually, the fact that all cell behaviours are expressed in terms of an energetic cost makes the addition of new model features relatively straightforward. The availability of a widely used software implementation in CompuCell3D is a further benefit in itself – it enables easy model sharing and collaboration between different groups and research fields, and the availability of a GUI for model setup opens the software to non-expert programmers.

A particular problem with the CPM is relating the energy constraints to measurable experimental parameters in order to make quantitative comparisons or predictions. Furthermore, it is not clear how a MCS is related to real time, and when a CPM is coupled to other models, such as a reaction-diffusion system for nutrient transport, the relationship between time and MCS must be estimated by comparing the observed model cell dynamics with experiment.

2.2 Off-lattice models

While the CPM improves the representation of cell shape over cellular automata and allows many mechanisms to be incorporated through the Hamiltonian, as noted above in most cases it is not possible to relate model parameters to experimentally measurable quantities. By removing the restriction that cells lie on a lattice, mechanical parameters, and hence cell motion can be characterised by experimentally measurable values, and hence quantitative predictions may, in principle, be made. The approaches that have been taken in off-lattice modelling may be broadly split into three categories: cell centre models, where cell location is characterised by a point in space; vertex dynamics models, where cell location is characterised by the location of the vertices of its boundary; and subcellular element models that divide the cell into smaller mesoscopic units. The mechanical aspects of off-lattice models are evolved by reducing the energy in the system, either using a Metropolis-Hastings approach with an energy function, similar to the CPM or by considering the total force $\mathbf{F}_i$ on a cell or vertex i and assuming over-damped motion to give a first-order equation of motion of the form

$$\eta_i \frac{d\mathbf{r}_i}{dt} = \mathbf{F}_i,$$

where $\mathbf{r}_i$ is its location, η_i is a damping constant for each cell or vertex, and t is time. We consider each of the modelling approaches in turn, and refer to the methods used to evolve them where appropriate.

2.2.1 Cell centre models

Cell centre models define each cell i by a single point in space $\mathbf{r}_i$, which may be assumed to be an arbitrary point in the cell, such as the centre of the nucleus. Typically, the spatial extent of the cell has either been modelled using a polygonal (polyhedral in 3D) region surrounding the cell, or as a fixed shape, such as a sphere characterised by a radius R_i . In tissues, cells exert mechanical forces on their surroundings, and to capture this a cell centre model also defines which of the cells are neighbours. Many variants of the cell centre approach have been considered, which differ in: their definition of the cell geometry and cell connectivity; how the cells interact with their surrounding cells and medium; and how processes at the tissue-level (nutrient or morphogen concentration) and processes at the subcellular level (cell growth and mitosis) are modelled and coupled together. We consider how these aspects have been approached for a range of examples from the literature.

Cell shape and neighbourhood

Based on the observations of Honda [1978] that cell shape in closely packed epithelial tissue and cultured monolayers may be approximately described by Dirichlet domains, a common approach is to compute the Dirichlet domains corresponding to the model cell centres (also referred to as the Voronoi tessellation). For a set of points $\{\mathbf{r}_i\}$, the Dirichlet domain corresponding to the point $\mathbf{r}_i$ is the region $\{\mathbf{x} \in \mathbb{R}^d : \|\mathbf{x} - \mathbf{r}_i\| < \|\mathbf{x} - \mathbf{r}_j\| \ \forall i \neq j\}$ where d is the dimension of the space. This not only gives a definition of cell geometry, but also of cell neighbours as those cells that share a non-zero boundary. The Dirichlet approach has been used in cell centre models of the sorting of cells driven by differential adhesion [Sulsky et al., 1984] and the spatio-temporal organisation of the colorectal crypt [Meineke et al., 2001, van Leeuwen et al., 2009]. Schaller and Meyer-Hermann [2005] used a 3D Dirichlet representation of cells in multi-cellular tumour spheroids, while Dunn et al. [2012a] and Dunn et al. [2012b] used a Dirichlet approach in a cross-sectional model of the colorectal crypt and epithelial layers respectively.

Although Dirichlet domains provide a good approximation of cell shape and neighbourhood structure in tightly packed populations, the model is not appropriate if cells are not clustered together, or may detach from a cluster. Furthermore, using the definition above, cells on the boundary of a population have infinite volume, which either necessitates the introduction of further ‘ghost centres’ surrounding the population, which do not correspond to model cells but act to make the area of all real cells finite [Dunn et al., 2012b]; or the hybrid approach used by Schaller and Meyer-Hermann [2005], where cells in isolation are assumed to take on a resting spherical shape, and cell-cell contact area is given by the minimum of the contact area defined by the Dirichlet domains and the contact area of two inter-penetrating spheres.

These problems are not present in other cell centre models, which typically consider a single deformable shape for cells: either a deformable sphere [Drasdo et al., 1995, Drasdo, 2000, Drasdo and Loeffler, 2001, Drasdo and Höhme, 2005, Galle et al., 2005, Höhme and Drasdo, 2010b, Buske et al., 2011, 2012, Macklin et al., 2012, Drasdo and Höhme, 2012, Frascoli et al., 2013] characterised by a variable cell radius R_i , or a deformable ellipsoid [Dallon and Othmer, 2004, Palsson and Othmer, 2000] characterised by three variable principle axes. These resting shapes may be deformed as a result of interactions with neighbouring cells and other extra-cellular media. In this approach, two cells i and j are typically considered to be neighbours if the overlap of their shapes, defined for spheres as $\delta_{ij} = R_i + R_j - \|\mathbf{r}_i - \mathbf{r}_j\|$, is above a certain (typically negative) threshold.

Cell-cell interactions

In cell centre models, the physical interactions between cells determine how the position of each cell changes over time. Interactions between cells typically represent the resistance to deformation and compression of the cytoskeleton as two cells are pushed together (hereafter referred to as repulsion), and the attraction, as a result of adhesive ligands on the cell membrane, of two cells being pulled apart. These properties are either characterised by assigning an energetic contribution from cell-cell interactions to an overall free energy

for the model population, or by defining force laws between pairs of cells that are summed to produce a resultant force on each cell.

The energetic approach was used in combination with a Dirichlet cell shape model by Sulsky et al. [1984], who defined a total energy $V = \sum l_{ij}e_{ij}$, where e_{ij} is the energy per unit length of boundary between cell i and cell j , and l_{ij} is the length of their shared boundary. By solving a variational problem, subject to the additional constraint that cells are incompressible, the sorting and engulfment of different cell types as a result of differential adhesion (characterised by e_{ij}) was reproduced. An energy-based approach was also used in the early work of Drasdo and co-workers [Drasdo et al., 1995, Drasdo, 2000, Drasdo and Höhme, 2005]. Drasdo et al. [1995] use a rudimentary interaction energy between two cells which is zero for sufficiently separated cells, takes a constant value for cells whose spherical bodies overlap, and is infinite below a threshold separation to represent the limited deformability of cells. In later work, Drasdo [2000] use an interaction energy that is proportional to the square of the separation between the cells, in addition to a bending and rotational energy, for single layered growing tissues. In the context of modelling tissue monolayers and three-dimensional cell aggregates, Drasdo and Höhme [2005] use the Johnson-Kendall-Roberts model of the interaction between two sticky elastic spheres [Chu et al., 2005]. In each case, the authors use a Metropolis-Hastings [Metropolis et al., 1953] approach, described previously for the CPM, to evolve the simulation by trialling small translations, rotations and growth increments of each cell, which are accepted with a probability depending on their effect on the total energy.

A more common approach to defining cell-cell interactions is as a ‘force law’ between pairs of cell centres. By summing all forces acting on each cell centre, and assuming overdamped dynamics, a system of ODEs of the form $\eta_i \dot{\mathbf{r}}_i(t) = \mathbf{F}_i(t)$ is numerically solved, where $\mathbf{r}_i$ is the location of cell i , $\mathbf{F}_i$ is the resultant force acting on cell i and η_i is a viscous drag term representing the effects of cell-cell and cell-medium friction. In the description below we define $\mathbf{r}_{ij} = \mathbf{r}_j - \mathbf{r}_i$, the vector joining the centre of two cells i and j , $\hat{\mathbf{r}}_{ij}$ the corresponding unit vector, and the force $\mathbf{F}_{ij} = F_{ij}\hat{\mathbf{r}}_{ij}$, where F_{ij} is the force magnitude.

In the Dirichlet domain model of Meineke et al. [2001] and later van Leeuwen et al. [2009], and in the deformable spheres model of one-layered tissues of Drasdo and Loeffler [2001], adhesive and repulsive interactions are lumped into a single linear force law of the form $F_{ij} = -k(|\mathbf{r}_{ij}| - l)$ where k is a constant and l is the equilibrium distance between cell centres. This force law leads to a larger attraction between cells as they become increasingly separated, which is unrealistic since in reality cells will eventually lose contact with one another as they are drawn apart rather than experiencing an increasing attractive force. As noted by Pathmanathan et al. [2009], this force law can lead to instabilities where three co-linear cells may all be pairwise attracting, with a larger force between the cells with greater separation. In the Dirichlet domain approach this is not problematic since, in tightly packed populations, neighbouring cells defined by the Delaunay triangulation are not separated by large distances. Furthermore, van Leeuwen et al. [2009] consider the spring constant k to be proportional to the length of the shared edge between two cells, leading to a smaller attractive force for cells that are sufficiently separated. In a deformable spheres approach it is normal to define a fixed interaction cut-off, beyond which the force is zero, representing the complete breaking of adhesive bonds between cells.

While a linear force law provides a simple and computationally efficient definition, by treating repulsion and adhesion as homogeneous it ignores the different mechanisms through which these forces arise in reality. Particularly in models where adhesion and deformation are coupled to subcellular models of growth and proliferation, rather than just governing spatial organisation, a more realistic description is required.

Schaller and Meyer-Hermann [2005] and Galle et al. [2005] decompose the force law into an adhesion, a deformation and a compression component. In both models, the repulsive force between two cells as a result of deformation of their cytoskeleton is given by the Hertz model where the interaction force is given by

$$F_{ij} = \frac{4E_{ij}}{3\sqrt{R_{ij}}} \delta_{ij}^{\frac{3}{2}},$$

where

$$E_{ij} = \left(\frac{1 - \nu_i^2}{E_i} + \frac{1 - \nu_j^2}{E_j} \right)^{-1},$$

and δ_{ij} is the overlap of the cell volumes, E_i and E_j are the Young's moduli and ν_i and ν_j are the Poisson ratios of each cell and $R_{ij} = (R_i^{-1} + R_j^{-1})^{-1}$ is the geometric mean of the cell radii. The extension of the Hertz model to include an adhesion component differs between the models of Galle et al. [2005] and Schaller and Meyer-Hermann [2005]: Galle et al. [2005] assume that adhesion energy (which is differentiated to obtain a force) is proportional to the surface area shared by cells in contact, while Schaller and Meyer-Hermann [2005] assume the adhesion force to be proportional to the surface area shared by cells in contact. In addition to a deformation and adhesion component, Galle et al. [2005] include a contribution characterising resistance of individual cells to changes in their volume by allowing each cell's radius to change dynamically when two or more cells are compressed together. Consequently, cells are assigned target volumes V_i^T , and an energetic cost given by

$$\frac{K}{2V_i^T} (V_i^T - V_i)^2, \tag{2.1}$$

is associated with deviation from this value, where K is a bulk modulus parameter.

The assumption made in the models of Schaller and Meyer-Hermann [2005] and Galle et al. [2005] that cell-cell adhesion is proportional to instantaneous contact area ignores the history of the interaction, which can be argued to play a role in cell-cell adhesion. Cells that are initially not touching ($\delta_{ij} < 0$) will share no adhesive bonds until they come in to contact ($\delta_{ij} = 0$). As they are forced closer together ($\delta_{ij} > 0$) their shared surface area will increase, and consequently the number of adhesive bonds between them will increase. If the cells are subsequently pulled apart the separation between their centres when they detach will be greater than the separation when they first touched ($\delta_{ij} < 0$) as a result of the adhesive bonds deforming the cell. This hysteresis effect is captured in

the Johnson-Kendall-Roberts model [Chu et al., 2005] which is used in the recent work of Drasdo and co-workers [Drasdo and Höhme, 2005, Höhme and Drasdo, 2010a, Drasdo and Höhme, 2012]. The Johnson-Kendall-Roberts force is given by

$$F_{ij} = \frac{4E_{ij}}{3R_{ij}}a^3 - (8\pi\sigma E_{ij}a^3)^{\frac{1}{2}},$$

where the parameter a is defined implicitly by the relationship

$$\delta_{ij} = \frac{a^2}{R_{ij}} - \left(\frac{2\pi\sigma a}{E_{ij}} \right)^{\frac{1}{2}},$$

and σ is an adhesion parameter. Hence F_{ij} is multivalued and depends on whether the spheres share a contact surface or not.

Pathmanathan et al. [2009] carried out a computational comparison of the linear, Hertz and Johnson-Kendall-Roberts force laws by measuring the response of a tissue to compression, tension and shear forces. Except for the instability of the linear law previously mentioned, no significant difference was observed in the bulk properties of tissues where cell-cell interactions are governed by the different laws. However, while the bulk passive properties of a tissue may be robust to different interaction laws, it is not clear whether simpler force laws may replace more detailed models such as the Hertz or Johnson-Kendall-Roberts approach when the cell cycle model is coupled to the adhesive and compressive state of the cell. For example, in the work of Galle et al. [2005], the growth and proliferation of each cell is governed both by its level of adhesion to an underlying substrate and the level of compression it experiences from neighbouring cells. In such examples, the use of a simpler force law may significantly alter the results.

Recently, cell centre models have been proposed that include further detail in the way cell-cell interactions and cell-basement-membrane interactions are modelled. For example, Frascoli et al. [2013] consider a non-uniform distribution of adhesive sites on the cell in the context of collective motion of cell pairs. In extensions of the model of Galle et al. [2005], the basement membrane has been modelled as either a fixed [Buske et al.,

2011] or deformable [Buske et al., 2012] triangular mesh where the mesh points interact with cell centres through an additional force law in order to model the organisation and development of the colorectal crypt. Other approaches consider the basement membrane as a fixed boundary condition [Macklin et al., 2012], or as an implicit structure that gives rise to a basement membrane force which stabilises a flat epithelial layer [Dunn et al., 2012b].

Nutrient transport

In studies where the spatio-temporal distribution of nutrients or morphogens plays a role in governing the proliferation and death of cells in tumour development [Schaller and Meyer-Hermann, 2005, Drasdo and Höhme, 2005, Höhme and Drasdo, 2010b, Macklin et al., 2012], or in the regulation of cell differentiation [van Leeuwen et al., 2009, Buske et al., 2011] in the colorectal crypt, it is necessary to model their transport and consumption explicitly. Most such cell centre models share a common approach in using reaction-diffusion PDEs of the form

$$\frac{\partial u(\mathbf{x}, t)}{\partial t} = \nabla \cdot (D(\mathbf{x}, t) \nabla u(\mathbf{x}, t)) - r(\mathbf{x}, t), \quad (2.2)$$

where $u(\mathbf{x}, t)$ is the nutrient concentration at location $\mathbf{x}$ and time t , $D(\mathbf{x}, t)$ is the diffusion coefficient (typically assumed constant), and $r(\mathbf{x}, t)$ is a source term for the production or consumption of the nutrient, which typically depends on the local cell density. To model nutrient transport in cases where temporal variation in the concentration of the substance is small, a constant distribution may be used, such as the model of local activity of the Wnt signalling pathway in the crypt developed by van Leeuwen et al. [2009] and also used by Buske et al. [2011].

Cell growth, proliferation and death

Models of growing tissues must include some representation of the cell cycle: the process through which every living cell regulates division. The eukaryotic cell cycle is a highly

regulated, complex process of growth, DNA replication and division, but it may be split into five phases [Alberts et al., 2002]. During the first of two ‘gap’ phases, G_1 , a cell grows and monitors its environment to ensure conditions are favourable for division. Subsequently, during S phase, DNA is replicated before a further ‘gap’ phase, G_2 , allows for further assessment of environmental conditions. A cell then completes the mitotic M phase during which it divides into two daughters. At any stage during the gap phases when environmental conditions become unfavourable for division the cell may enter a non-proliferating, resting state called quiescence or G_0 phase.

In the off-lattice models considered here, the cell cycle is typically incorporated as a simple rule-based progression of each cell through the various phases outlined above, culminating in a division event when an additional cell is inserted into the population. In the Dirichlet domain model of Meineke et al. [2001] for example, division of a cell is assumed to occur after a period of time that is chosen randomly in the range 10-15 hours depending on the cell type. In their extension of this model, van Leeuwen et al. [2009] further assume that a cell only enters the cell cycle if there is a sufficient level of signalling factors associated with the Wnt pathway available - otherwise the cell terminally differentiates and does not divide again.

For deformable spheres models, the favoured approach is to assume that the cell grows and doubles its volume during interphase ($G_1 + S + G_2$) before splitting into two smaller daughters of half the volume of the parent during M phase. In models where the cell radius R_i is not a dynamic variable of the model that adapts to the local environment, the radius is assumed to grow in small increments until the cell volume has been doubled [Drasdo, 2000, Drasdo and Höehme, 2012, Höehme and Drasdo, 2010b, Buske et al., 2011, Schaller and Meyer-Hermann, 2005]. To model contact inhibition of growth, where a cell enters quiescence when it becomes sufficiently deformed or compressed, the growth in the radius is paused if any neighbouring cell centre comes within a critical threshold distance of the cell. In contrast, in the model of Galle et al. [2005], where the cell radius varies in response to local compression, it is the target volume of the cell, given by V_i^T in equation (2.1),

which is increased, leading to a subsequent increase in the cell volume. If the ratio of the actual volume to the target volume of a cell falls below a critical threshold, the growth of the cell is paused until such time as the volume recovers above the threshold.

As well as being regulated by deformation, the cell may arrest its growth phase if the availability of nutrients is not sufficient. Macklin et al. [2012] assume a cell enters a proliferative state with a probability dependent on the local oxygen concentration, while Drasdo and Höhme [2005] consider local glucose concentration as a limiting factor, with a cell only entering the cell cycle if the local concentration exceeds a fixed threshold.

When a cell divides, the new cell centre must be accommodated in the population. A common approach is to preserve the centre of mass of the parent cell and place the two daughters a small distance apart along a mitotic axis vector that passes through the mother cell's centre of mass [Macklin et al., 2012, Galle et al., 2005, Schaller and Meyer-Hermann, 2005, Höhme and Drasdo, 2010b]. Meineke et al. [2001] differ by leaving one cell fixed, and placing a daughter cell in close proximity, while models of growth in epithelial layers align the mitotic axis vector parallel to the plane of the epithelial layer [Dunn et al., 2012a,b, Drasdo, 2000, Drasdo and Loeffler, 2001] or the substrate [Galle et al., 2005].

In addition to cell growth and division, tissue size and composition are affected by cell death [Alberts et al., 2002]. In the examples reviewed here, two forms of cell death are considered – anoikis and necrosis. Anoikis is a form of programmed cell death that occurs when an epithelial cell loses sufficient contact with the basement membrane or substrate on which it is growing [Frisch and Francis, 1994] and is implemented by either removing model cells instantly [Dunn et al., 2012a,b] or transiently [Galle et al., 2005, Buske et al., 2011, Macklin et al., 2012] when contact is lost.

In models of early tumour growth, where a vascular system is not fully developed, a lack of nutrients may also lead to necrosis, which is an uncontrolled form of cell death [Freyer and Sutherland, 1986]. In Macklin et al. [2012], cells become necrotic with a probability dependent on the local oxygen concentration, while Drasdo and Höhme [2005] and Höhme

and Drasdo [2010b] consider glucose to be the key nutrient in necrosis. Schaller and Meyer-Hermann [2005] include the influence of both oxygen and glucose. In all these models, necrotic cells are removed transiently at a fixed rate to model the gradual degradation of the cellular material.

Available software

Chaste [Pitt-Francis et al., 2008] is the main open-source, extensible software library for cell centre modelling of cell populations. While it does not offer the graphical interface for users provided by software such as CompuCell3D, it does offer a broad choice of modelling approaches - each of which may be modified by a user for their own purposes, given some basic knowledge of the C++ programming language. The Dirichlet domain approach to cell geometry can be used in 2D [van Leeuwen et al., 2009, Dunn et al., 2012a,b] or 3D [Mirams et al., 2013], as can an overlapping spheres approach [Pathmanathan et al., 2009, Mirams et al., 2013]. Cell-cell mechanical interactions may be written as simple C++ classes and included into the models, and implementations of the linear spring law, and the modified Hertz law used by Buske et al. [2011] are currently available. At the cellular level, simple stochastic length cell cycle models may be interchanged with more detailed models that include regulation by contact inhibition, by local levels of diffusing nutrient or by signalling factors secreted by neighbouring cells. The software also allows for coupling PDEs for modelling nutrient transport to the cell-level model.

Höehme and Drasdo [2010a] provide a software implementation of their deformable spheres model [Höehme and Drasdo, 2010b]. While the code is not open-source, and hence not extensible, a GUI is provided for simple visualisation and interaction with a simulation and all of the model parameters may be altered. However, its application is restricted to simulating the growth of 2D monolayers and 3D tumour spheroids.

Open-source implementations of centre-based models exist for other specific applications beyond the scope of this review. For example, in the context of growing microbial biofilms, iDynoMiCS [Lardon et al., 2011] provides an open-source modelling framework

based upon a cell centre approach.

2.2.2 Vertex dynamics models

Although the Dirichlet domain approach outlined above provides a good approximation of the polygonal shape of epithelial cells in closely packed populations, the fact that the shape is generated from cell centres is not ideally suited for experimental comparison. The vertex dynamics model, however, explicitly represents the junctions between distinct cells, which are more easily observed in experimental imaging [Nagai and Honda, 2001]. We briefly describe a 2D instance of the model developed by Farhadifar et al. [2007], and then consider extensions to 3D.

Each cell is described by a set of vertices that form a polygonal tessellation of the domain space. Each vertex i , located at $\mathbf{r}_i$, forms the junction of three cell edges. The motion of the vertices is assumed to be over-damped, resulting in a force balance between friction characterised by a damping constant η_i and cell boundary tension, cellular resistance to deformation and contractility of the cell perimeter. Consequently the equation of motion for each vertex is

$$\eta_i \frac{d\mathbf{r}_i}{dt} = -\nabla U,$$

where $U = U_A + U_T + U_C$ is a potential function encompassing the resistance of a cell to deformation from its target area U_A , line tensions at junctions between cells U_T and the contractility of the cell perimeter as a result of the actin-myosin ring at the apical cell surface U_C . U_A is given by

$$U_A = \sum_{\alpha} \frac{K_{\alpha}}{2} (A_{\alpha} - A_{\alpha}^0)^2, \quad (2.3)$$

where K_{α} is an elastic coefficient for cell α , A_{α} is its present area and A_{α}^0 its preferred area. The line tension at junctions between cells is characterised by

$$U_T = \sum_{\langle ij \rangle} \Lambda_{ij} ||\mathbf{r}_i - \mathbf{r}_j||,$$

where Λ_{ij} is the magnitude of the tension along the line between vertices i and j and the sum is over all connected vertices. Finally, U_C is given by

$$U_C = \sum_{\alpha} \frac{\Gamma_{\alpha}}{2} L_{\alpha}^2,$$

where L_{α} is the perimeter of cell α and Γ_{α} is a contractility parameter. Further components may be included into U to include, for example, boundary surfaces, or additional forces applied to the population. An edge switching mechanism is also included to prevent the collapse of two vertices together and enable changes in the cell neighbourhood structure. This model was able to account both qualitatively and quantitatively for the observed packing of cells in the *Drosophila* wing disc epithelium in terms of cell area and neighbourhood topology [Farhadifar et al., 2007]. The vertex dynamics model extends naturally into 3D by treating each cell as a convex polyhedron, and each vertex as the intersection of four cell edges [Honda et al., 2004].

Similarly to the lattice-based and cell centre models already considered, the basic vertex dynamics model has been extended to include cell growth, proliferation and coupling with continuum field descriptions of extracellular solutes. For example, Smith et al. [2012] use a 2D vertex dynamics model of a growing epithelial layer, coupled to a continuum description of morphogen transport, to model development of the *Drosophila* imaginal wing disc. Cell growth, with implicit contact inhibition, is included in the vertex model by a transient increase in the cell target area (given by A_{α}^0 in equation (2.3)) with a cell then dividing in half along a random mitotic axis into two daughters once it is fully matured. The transport of a morphogen believed to regulate growth in this system is modelled by a reaction-diffusion PDE of the form in equation (2.2), which is numerically solved using the vertices of the cell population and the cell centres as generating points for a finite element mesh. The different scales are coupled by assuming that the growth

rate of the cell target area depends linearly on the local morphogen concentration.

While there is a lack of comprehensive comparisons between vertex models and their cell centre counterparts, Osborne et al. [2010] found models of homeostatic colorectal crypt dynamics, and the invasion of a mutant cell type in the crypt, to be robust to the choice between a vertex dynamics and a cell centre Dirichlet domain modelling approach. To our knowledge, Chaste provides the only open-source software environment for vertex dynamics model simulations [Mirams et al., 2013].

2.2.3 Agent-based models

The cell centre and vertex dynamics models were initially developed from the perspective of modelling physical cell-cell interactions, such as adhesion and repulsion, and later extended to include a rule-based model of the cell cycle, a so-called ‘middle-out’ approach. A further class of model, referred to as *agent-based* models, focus on processes at the cellular level and treats cells specifically as state machines, with transitions between different subcellular states occurring as a result of explicit rules for each cell. Cells are self-contained units and may not access the state variables of other cells directly. Instead, communication between cells occurs through a message board. Cells may read from the message board before executing a rule, and write to the message board after executing a rule. Since these models focus on cell division and growth they are typically considered over longer time scales than the cell centre models described above, and advance the model in incremental time steps of approximately 30 minutes. Therefore, the cells are assumed to be in mechanical equilibrium, and a force balance in accordance with a simple adhesive-repulsive force law is achieved using an iterative numerical technique, rather than integrating over-damped equations of motion. This enables computational studies that extend to the development of cell populations on the scale of years [Li et al., 2013]. To provide an overview of this modelling approach, we shall describe an agent-based model of epithelial growth and repair, initially developed by Walker et al. [2004], and then consider further extension and development of this model.

The model considers a population of 3D ellipsoidal cells growing on a 2D substrate, with each cell realised as a software agent with a set of state variables. A simulation is executed by iteratively updating each cell's state variables (representing, for example, its progress through the cell cycle) based on a set of rules that define the transition from one state to another. The rules may incorporate data from the current state of the cell as well as the cell's environment. Once these update rules have been applied to every cell in turn, a single iteration, or 'tick' is complete, representing about 30 minutes of real time. A numerical equilibration process is then carried out to eliminate physical overlaps between cells.

This model has been used to study the growth characteristics of monolayers of epithelial cells in different physiological conditions and compared with growth cultures *in vitro*. Walker et al. [2004], in their original model, looked at the different growth patterns obtained in monolayer cultures of epithelial cells depending on the level of extra-cellular calcium. Since the formation of inter-cellular bonds depends on calcium concentration, it was found that cells seeded in relatively high physiological calcium conditions formed adhesive bonds with neighbouring cells, preventing migration. At the population scale this emerges as growth of the population in colonies, compared to a more confluent, dispersed growth pattern in low calcium conditions, in agreement with *in vitro* observations.

The early model only considered physical cell-cell interactions through adhesive bonds and volume exclusion, however the nature of the model allows for more detailed models of subcellular processes to replace the simple initial rules. Walker et al. [2006] extend the model to consider molecular-level signaling through the epidermal growth factor receptor. More detailed intracellular pathways models are included by Walker et al. [2008], and Walker et al. [2010] model interactions of a normal and mutant population. Further developments have included the integration of detailed subcellular models of gene expression and signalling, to replace the rules in earlier models. Sun et al. [2009] and Adra et al. [2010] develop a 3D extension to the model that includes subcellular modelling of the transformation growth factor TGF- β 1. Recently, Li et al. [2013] have used this extended

model to study the long-term behaviour of clonal populations in skin epithelia to compare different hypotheses of stem cell clone survival.

The principal difference between the agent-based model and the cell-centre models described in Section 2.2.1 is the approach to modelling cell-cell physical interactions. While the cell-centre models typically solve an overdamped first-order equation of motion for each cell based on a force balance, the agent based model assumes that over the time-step used in simulation (approximately 30 minutes) the cells will be in mechanical equilibrium, and use an iterative numerical scheme to reduce overlap between cells.

Although initially developed as a MATLAB [MATLAB, 2010] program, the agent-based model described above may be simulated in the FLexible Agent-based Modelling Environment (FLAME) [Coakley et al., 2013]. This environment, rather than being a specific model implementation, generates simulation code based on a definition of agent types and state transition rules. By enforcing that all agents are state machines which communicate indirectly with each other through message boards, the software can extract a dependency graph for the different rules and communications upon which the rules depend, enabling efficient scheduling of agent updates. Furthermore, since agents do not access each other's state variables directly, but rather through the message board, the software may generate efficient parallel code for use on High-Performance Computing (HPC) [Coakley et al., 2012] or General Purpose Graphics Processing Unit (GPGPU) technology [Richmond et al., 2010].

2.2.4 Subcellular element models

The off-lattice models considered so far treat the cell as the fundamental unit of the model, and its geometry as a (deformable) model input, either as a polyhedron, a sphere or an ellipsoid. Furthermore, the cell has been treated as a single biomechanical unit, with forces acting through its centre or a small number of boundary vertices. Modelling individual cells at a finer resolution by considering them to be composed of mesoscopic volume elements enables cell geometry and mechanical response to be an emergent, rather

than imposed, property of the model. We consider two such models here which are natural extensions of the cell centre and vertex dynamics approaches respectively, and have been used to model populations of cells.

The ScEM was initially developed by Newman [2005] to model the adaptive shape and viscoelastic properties of biological cells. In the ScEM, a single cell is divided into mesoscopic volume elements ('elements'), of approximately $1\mu\text{m}^3$. The elements each contain a network of cytoskeletal filaments which are intricately connected to the filaments of neighbouring volumes. In the model this is approximated by considering the centres of mass of neighbouring elements to be subject to an attractive-repulsive force, similar to those considered in Section 2.2.1 for cell centre models. We omit the model details here as they are considered explicitly in Chapter 6.

By carrying out *in silico* bulk rheology experiments on a single cell on intermediate time scales (10 s), Sandersius and Newman [2008] show that it is possible to scale the model parameters so that the biomechanical properties of an ScEM cell are independent of the number of elements of which it is composed. The bulk rheology experiments follow a creep-stress protocol similar to those used *in vitro*, where a constant extensile force is applied to the surface of a cell, while the opposite surface is fixed, before the extensile force is released. The strain on the cell is measured as the extension of the cell in the direction of the force, relative to its initial linear size. The authors observe that for a given applied force the strain experienced by a cell is finite and plateaus after around 1 s, and that the cell recovers its original shape completely after the force is released. Their results are in quantitative agreement with the *in vitro* experiment of Wottawah et al. [2005] and consequently suggest that the ScEM is able to capture the passive biomechanical properties of biological cells.

However, on timescales longer than a few seconds, the cell is not a passive material, but responds actively to stresses placed upon it. To incorporate the biomechanical adaption of cells to their environment, Sandersius et al. [2011] extend this basic model to include active cell biomechanical processes. By 'fading-in' (inserting over short time scales) and

‘fading-out’ (removing over short time scales) subcellular elements of a cell in response to different stimuli, they are able to recreate active cytoskeletal remodelling, measure epithelial tissue viscosity as a function of subcellular remodelling and recreate streaming patterns of chemotactic cells in a tissue monolayer. In particular, cytoskeletal remodelling was achieved by fading-in elements in regions of high stress in the cell, and fading-out in regions of low stress. Repeating the cell rheology experiments of Sandersius and Newman [2008], they show that, rather than reaching a deformation plateau, the active processes make cells fluid-like on longer timescales (100 s) for large applied forces, and, furthermore, for smaller applied forces the remodelling acts to make the cell contract.

As with other individual-based models, the ScEM has been extended and coupled to models at different scales. For example, Christley et al. [2010] developed a model of epidermal growth on a basal membrane. In addition to the mechanical properties of the cell from the ScEM the authors include a simple growth and division algorithm, where a cell grows by repeatedly adding elements to its volume, and divides by splitting its elements between two daughters. Furthermore, the growth in this model is coupled to a subcellular gene network incorporating intercellular Notch signalling. The model is presented in the context of developing an algorithm for simulating the ScEM on GPGPU technology, and source code is provided for the simulations. However, this implementation is restricted to the model considered, and does not provide the same flexible modelling environments seen in CompuCell3D, Chaste or FLAME.

In addition to extensions to include mitosis, the model has also been coupled to a fluid flow model to simulate the attachment of platelets (modelled using the ScEM) to damaged blood vessel walls in thrombus development [Sweet et al., 2011]. By using a detailed mechanical model of the platelets the authors uncovered a dependence on the platelet stiffness to how it moves and rotates in the fluid and, consequently, how it attaches to injured sites on the vessel wall.

While the ScEM is a natural extension of the cell centre models, the vertex dynamics model has also been refined to consider cells as being composed of mesoscopic units in

the finite element model of Brodland and Chen [2000a]. In this model, cell shape is analogous to that in the vertex dynamics model, in that it is polygonal and defined by the location of its vertices. Additionally, each cell polygon with n edges is further decomposed into n triangular ‘finite elements’, joined at the centroid of the polygon. Similarly to the vertex dynamics model, the motion of cell vertices is driven by interfacial tension between cells of the same and different types. The model is used to test and propose an alternative to the differential adhesion hypothesis of Steinberg [1964], namely that cell sorting and engulfment is not driven by differential adhesion but by differential surface contraction [Brodland and Chen, 2000b]. The model was also later extended to include cell mitosis [Brodland and Veldhuis, 2002] using an approach similar to the vertex dynamics model where cells grow, and divide by inserting new boundary vertices along a mitotic axis.

2.3 Continuum approximation of discrete models

One approach to overcoming the computational cost of simulating a large number of cells using an individual-based approach is to use a continuum approximation of the discrete model. A continuum model can either be used in a hybrid approach to model regions of the population that may be considered to be homogeneous, or the mathematical technique of coarse-graining may be applied to convert an individual-based model to an equivalent continuum model.

For applications where it is only necessary to study individual cells in detail in a small region of the domain, the remaining cells or extra-cellular media may be modelled as a continuum in a hybrid approach. For example, Kim et al. [2007] developed a hybrid model of a growing avascular tumour spheroid. As a tumour spheroid grows *in vitro*, cells at the core enter a quiescent (non-proliferative) state as a result of nutrient depletion and mechanical stress, and eventually enter necrosis and die. As a result, these cells do not contribute to the overall growth of the spheroid. The authors therefore assume that cells

in these necrotic and quiescent regions may be adequately represented by a continuum model of a linear viscoelastic solid. Only cells in the proliferative rim of the spheroid are treated individually using an adapted deformable ellipsoid model of Dallon and Othmer [2004]. Cell growth and division is assumed to be limited by both mechanical stress and nutrient availability. The transfer of force between the continuum and individual-based models is handled by adapting the boundary conditions for each model based on the other. Furthermore, individually modelled cells may be subsumed into the continuum region if they become sufficiently mechanically stressed or nutrient deprived. Using this approach, the number of individually modelled cells is limited to a few hundred even in a developed spheroid, which keeps the model computationally tractable, beyond what would be possible if every cell was modelled individually. In applications where it is suitable to consider only a small subset of cells individually, this approach allows a significant reduction in computational cost, whilst maintaining the key features of the individual-based modelling approach.

One of the difficulties with such a hybrid approach is how to appropriately blend the continuum and discrete models at their boundary. Along with computational cost, this is one of the motivating factors for recent work that attempts to relate individual-based models to an analogous continuum model using a coarse-graining approach. Initial work has focused on 1D models. Fozard et al. [2010] derived a continuum approximation of a 1D vertex dynamics model that excludes cell division and where cell-level parameters only vary slowly in space, or are periodic between cells. Similarly, Murray et al. [2009] have used coarse-graining to go from a discrete to a continuum model for a 1D cell centre approach with a simple linear force law. By adding non-linear source terms to the resulting diffusion PDE for cell number, the model can also accurately capture growth and cell division. This model was later shown to produce good agreement with a model of cell dynamics in the colorectal crypt [Murray et al., 2011]. While these approaches are restricted in their analysis to 1D models, Davit et al. [2013] have assessed the applicability of a coarse-graining rule from crystal mechanics to individual-based models of passive cell

populations (i.e. without cell growth and division). For small deformations, the authors found that the Cauchy-Born rule, which, in the context of cell populations, states that the cell level displacements may be mapped linearly from the tissue scale deformations, may be applied to 2D cell centre models. However, the rule is found not to hold for the vertex dynamics model. A continuum analogue of the 2D CPM with chemoattractant driven cell motion has also been derived [Lushnikov et al., 2008].

As stated, these coarse-graining approaches may inform the development of hybrid models. Furthermore, they may be used to aid parameterisation and classification of individual-based models. By moving to the continuum model, population-level characteristics may be able to be expressed in terms of parameters of the individual-based model, which may in turn be constrained by comparison with population-level experimental observation. For example, by relating non-linear force laws for cell centre models to diffusion coefficients for cell number in a continuum model, Murray et al. [2012] are able to form a bridge between measurable dynamics of cell number at the tissue scale and hypothesised force laws between individual cells in an individual-based model.

2.4 Summary

This review has shown how individual-based modelling of cell populations has developed and encompasses a wide range of approaches. The relatively simple cellular automata provide a computationally efficient approach in situations where cell shape and mechanics are not of importance. The CPM improves the representation of cell shape and may incorporate many observed cellular mechanisms in an illustrative manner through the potential function. Off-lattice models provide a more realistic representation of cell mechanics at the cost of less computational speed. Dirichlet domain cell centre and vertex dynamics models provide approximations of the polygonal shape of cells in densely packed populations, while deformable spheres and ellipsoids model changes in cells from their resting shape implicitly by allowing the volumes to overlap. By simulating over longer timescales

and assuming that cells are in mechanical equilibrium, agent-based models are able to better capture the long-term dynamics of a population and offer a further environment for studying the emergent effects of perturbations to single cells in an interacting population. Further resolution in terms of cell shape and mechanics may be obtained by using the ScEM model. In each case, off-lattice models represent cell-cell interactions as either short-range force laws or a potential energy function to model the elastic properties of the cytoskeleton and the adhesion of cell membranes of neighbouring cells.

In each of the cases considered, the original cell-level model has been extended to include detailed models of the subcellular machinery that governs individual cell behaviour, as well as the transport of extra-cellular solutes such as nutrients and growth factors. While each is a modelling field in its own right, the ability to ‘plug’ such models into single cells and the population respectively enables the study of the emergent effects at the tissue-level of feedback between the different scales of model.

Although such detailed multiscale models enable *in silico* testing of hypotheses and quantitative predictions, they present formidable barriers to entry to a modeller, in terms of the cost of software development as well as computational resources. We have seen that this has been addressed for the CPM by the development of CompuCell3D as a modelling environment. For off-lattice models, Chaste provides a comprehensive library of open-source code for modellers. FLAME offers a general agent based modelling tool, where simulation code for state machine models may be generated. Although CompuCell3D offers parallel functionality for the CPM and FLAME offers auto-generation of parallel code for state machine models, there is no parallelised open-source equivalent for the cell centre models described in Section 2.2.1.

Chapter 3

Overlapping spheres model of a cell population

In this chapter we describe the individual-based model of a population of cells for which we have developed and implemented a parallel simulation algorithm. This describes the model as already implemented in Chaste, and as such is not a novel development of this project. We also provide an overview of the approach used in the original serial implementation of the algorithm, and discuss the structure and development of the Chaste software library for cell population modelling.

3.1 Model details

We consider an off-lattice cell centre model where each cell i in a population of (variable) size N is associated with a point $\mathbf{r}_i$ in either 2D or 3D space for $i = 1, \dots, N$. We assume that each cell is modelled as a sphere of radius R_i centred on $\mathbf{r}_i$. Two cells i and j are assumed to be neighbours if their centres satisfy $\|\mathbf{r}_i - \mathbf{r}_j\| < r_c$ for a fixed constant r_c , where $r_c > R_i$ for all i .

Each cell is endowed with a cell cycle model, which governs its progress through the phases introduced in the last chapter, namely G₁, S, G₂ and M, as well as conditions for entry into the resting G₀ phase. For simplicity we assume that each phase has a

constant length, however the generality of the implementation and of our approach to parallelisation allows for the length of the respective phases to be governed by any of the cell state variables, or the state variables of neighbouring cells. For example, a contact inhibition version of the cell cycle model may be used, in which a cell enters the quiescent G_0 phase in response to the proximity of neighbouring cells. Upon completing G_2 phase, a cell (cell i) enters M phase and divides into two daughter cells. A random mitotic unit vector $\hat{\mathbf{m}}$ is chosen and the daughters cells are placed at $\mathbf{r}_i \pm s\hat{\mathbf{m}}$, where s is a constant division separation parameter.

A cell may be instructed to die, dependent on its state variables or those of its neighbouring cells. A cell instructed to die enters an apoptosis phase of fixed length duration, after which it is removed from the population.

A force-based approach is taken to governing cell-cell mechanical and adhesive interactions. Again, the generality of our approach allows for the definition of any force law between neighbouring cells (those that satisfy $\|\mathbf{r}_i - \mathbf{r}_j\| < r_c$). We use a simple adhesive-repulsive quasi-linear force law introduced by Davit et al. [2013], which is designed to overcome the instabilities of the linear spring law noted by Pathmanathan et al. [2009]. For a pair of neighbouring cells i and j , we define the force $\mathbf{F}_{ij}$ on cell i due to cell j to be

$$\mathbf{F}_{ij} = \begin{cases} k l \log(1 + \delta_{ij}/l) \hat{\mathbf{r}}_{ij}, & \text{if } \delta_{ij} \leq 0, \\ k \delta_{ij} \exp(-5\delta_{ij}/l) \hat{\mathbf{r}}_{ij}, & \text{otherwise,} \end{cases} \quad (3.1)$$

where $\hat{\mathbf{r}}_{ij}$ is the unit vector joining the cell centres, l is the equilibrium distance between the cells, $\delta_{ij} = \|\mathbf{r}_{ij}\| - l$ is the overlap between the cells and k is a parameter governing the strength of the interaction. For normal cells $l = R_i + R_j$, but for apoptotic cells l is shrunk linearly to zero during the apoptotic phase. The structure of Chaste, and of the algorithm we develop allows parameters in this formulation to depend on other cell and cell neighbour variables. We plot the signed magnitude of $\mathbf{F}_{ij}$ as a function of δ_{ij} in Figure 3.1, where it can be seen that this force gives rise to a strong repulsive force

as cells are compressed together, and a relatively weak attractive force for cells in close proximity. The total force $\mathbf{F}_i$ on a cell i is then given by

$$\mathbf{F}_i = \sum_j \mathbf{F}_{ij} + \mathbf{F}_{\text{ext}}, \quad \text{for } i = 1, \dots, N,$$

where $\mathbf{F}_{\text{ext}}$ may incorporate external hypothetical forces not due to cell-cell interactions, such as a random motion or chemoattractant force. The sum is taken over neighbouring cells j .

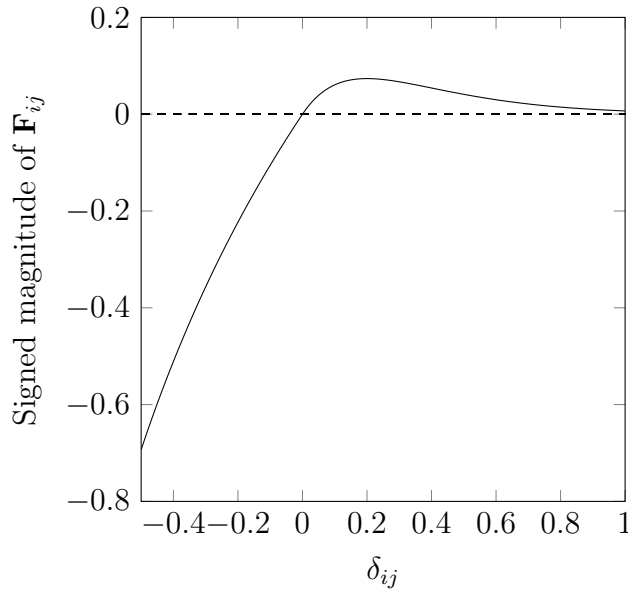


Figure 3.1: An example force curve for the simple interaction force between cell centres. The plot is shown for values $k = l = 1$. Negative values of the magnitude of $\mathbf{F}_{ij}$ represent repulsion, while positive values represent attraction.

The motion of each cell i in response to the net force $\mathbf{F}_i$ is assumed to be damped by the viscosity of the surrounding medium and friction with other cells. This is combined into a single viscous drag term so that, applying Newton's second law, we have

$$m_i \ddot{\mathbf{r}}_i = \mathbf{F}_i - \eta_i \dot{\mathbf{r}}_i, \quad \text{for } i = 1, \dots, N,$$

where m_i is the mass of cell i and η_i is a viscous drag parameter for cell i . Dallon and Othmer [2004] note that for a cell of linear size $10\mu\text{m}$ moving in water, the Reynolds

number (the ratio of inertial force to viscous forces) is of the order 10^{-8} to 10^{-6} , and is likely to be even smaller if the effects of surrounding cell cytoplasm are included. In low Reynolds number regimes, the viscous forces dominate the inertial forces, and we therefore make the assumption that the term $m_i \ddot{\mathbf{r}}_i$ may be neglected. Consequently the equation of motion for each cell is given by

$$\dot{\mathbf{r}}_i = \frac{1}{\eta_i} \mathbf{F}_i, \quad \text{for } i = 1, \dots, N.$$

3.2 Numerical solution of model equations

Each simulation is assigned an end time $T > 0$, and is progressed in fixed length time steps $0 < \Delta t \ll T$. Given the state of the cell population at time $0 \leq t < T$, the current time is first incremented to $t + \Delta t$. The cell cycle model is then updated to the correct phase based on the incremented time, and any environmental conditions. Any cells that are marked to divide or to die carry out this process by splitting into two daughters or being marked for deletion and then removed immediately, respectively.

The neighbours of each cell are then calculated, and the total instantaneous force $\mathbf{F}_i(t)$ on each cell is summed from its interactions with neighbouring cells, such as the force described in the previous section. The location of each cell is then updated synchronously (i.e. forces are not recalculated after updating each cell's location) using the forward-Euler scheme for an initial value (ODE) problem [Süli and Mayers, 2003]:

$$\mathbf{r}_i(t + \Delta t) = \mathbf{r}_i(t) + \frac{\Delta t}{\eta_i} \mathbf{F}_i(t), \quad i = 1, \dots, N. \quad (3.2)$$

The numerical solution of equation (3.2) is an approximation and therefore introduces an error. For a single cell, the error made in updating its location in a single time step is proportional to $(\Delta t)^2$. For inappropriately large values of Δt , this error can lead to instabilities in the motion of cells. Small values of Δt lead to increased computational costs. The value of Δt to choose depends on the model forces and the motion of the

cells. A common approach to choosing Δt is to ensure that halving the time step does not change the broad results of the simulation [Höehme and Drasdo, 2010b].

3.3 Software implementation

The model described in the previous sections has been implemented using a serial algorithm in the Chaste software library [Mirams et al., 2013]. In this section we describe the key features of Chaste and its development methodology as well as how the cell population simulation code is structured and implemented.

3.3.1 Chaste

The Chaste project began in 2005 as a collaborative effort to develop an open-source software resource of reusable code for modelling in biology. It currently offers functionality for individual-based cell population and cardiac electrophysiology modelling and simulation, with over 500,000 lines of code added by over 50 developers¹. Being written in the object-oriented programming language C++ [Stroustrup, 1997] brings a number of benefits. The class structure of the code means that functionality may be shared between the two modelling areas, such as mesh classes that are used for both numerically solving PDEs in cardiac modelling and representing the physical structure of a population of cells in cell-based modelling. Furthermore, being a compiled, rather than interpreted, language means that C++ typically outperforms languages such as MATLAB [MATLAB, 2010] in terms of execution speed.

Agile development

One of the novel features of Chaste is that it has been developed in accordance with agile software development principles [Martin et al., 2012]. While a full description of these principles and their application in Chaste is beyond the scope of this thesis, we briefly

¹Correct as of September 2013. Up to date statistics can be found at an independent code profiling site <http://www.ohloh.net/p/Chaste>.

outline the key methodologies that we have applied in this project. Further details on the agile approach in the context of developing scientific software in Chaste are reported by Pitt-Francis et al. [2008] and Mirams et al. [2013].

One of the realities of large software projects are ‘bugs’; that is, mistakes in the code that cause the software to behave in a faulty manner or not as intended. While some bugs cause obvious problems, such as failure of the code to compile, others can be subtle and not immediately obvious. In scientific computing software this can lead to erroneous results being produced and possibly incorrect conclusions being drawn. A key principle of agile development that has been used in the Chaste project is Test Driven Development (TDD). TDD specifies that ‘unit tests’ of functionality are written before the source code that implements it. The tests are small programs that are executed using the main source code, and either pass or fail depending on whether the software being tested performs as expected. As an example, Listing 3.1 shows a unit test that we wrote for a class that enables C++ objects to be sent between different processors in parallel. On line 3 we create an instance of the class we are testing. On lines 5-18 one processor creates some test objects and sends them to all other processors at line 16. On lines 19-30 all other processors use the communication object to receive the classes, and then a number of manual tests interrogate the received data to make sure it is correct.

```

1 void TestSendingClass() throw (Exception)
2 {
3     ObjectCommunicator<ClassOfSimpleVariables> communicator;
4
5     if (PetscTools::AmMaster())
6     {
7         // Create a simple class to send
8         ...
9         boost::shared_ptr<ClassOfSimpleVariables> p_new_class(
10             new ClassOfSimpleVariables(42,"hello",doubles,bools));
11
12         // Send the class
13         for (unsigned p=1; p < PetscTools::GetNumProcs(); p++)
14         {
15             // Arguments are object, destination, tag
16             communicator.SendObject(p_new_class, p, 123);
17         }
18     }
19     else
20     {
21         boost::shared_ptr<ClassOfSimpleVariables> p_recv_class;
22         // Receive class
23         p_recv_class = communicator.RecvObject(0, 123, status);
24
25         // Check that the values are correct
26         TS_ASSERT_EQUALS(p_recv_class->GetNumber(),42);
27         ...
28         TS_ASSERT_DELTA(p_recv_class->GetVectorOfDoubles()[0],1.1,1e-12);
29         ...
30     }
31 }

```

Listing 3.1: An example unit test of the `ObjectCommunicator` class. Some code is omitted for brevity, denoted by ‘...’. A simple class of variables is created and sent between processors, before testing that the correct data have been received.

While unit testing does not guarantee bug-free code, it has a number of benefits beyond simply avoiding mistakes. First, by writing the test before the class, the developer is forced to think about the interface of the class, that is, how it will be used in practice and how it will function with other classes in the software, leading to a better and more flexible code structure. Second, the collection of tests produced define the functionality of the entire code base. If the tests all pass, the software is considered to be ‘working’. This gives developers working in a team (or a research group) confidence to improve or change the structure of the code (called *refactoring*): provided they ensure that all tests still pass they can be sure that they have not accidentally broken the functionality of the code. Finally, the suite of tests developed act as a type of dynamic documentation. Although adding sufficient explanatory comments to the code is still necessary, comments can quickly become inconsistent or outdated; on the other hand the tests tell a new user exactly what a class can and should do, and how to use it.

In addition to running the unit tests for Chaste every time a change is made to the source code, a number of other tests are carried out daily on the most recent version.

These include: larger acceptance tests, which test the large-scale functionality of the classes in the library by running example simulations; memory testing, which ensures that no memory is leaked when the unit tests are carried out; and coverage tests which ensure that every line of source code is used in at least one of the tests. These additional tests further ensure the quality of the code and that the test-first structure is maintained.

Not every aspect of agile software development is appropriate or possible to follow in a research environment. For example, the agile methodology insists that code should be developed by pairs of programmers sitting at a single computer – one typing and the other thinking ahead and acting as a quality control. In a commercial software development team, where every developer’s sole aim is to implement the functionality required by the customer, this approach leads to high quality code. However, in most academic environments, the researcher assumes the position of both the ‘customer’ and ‘developer’ and each researcher will have their own goals and projects. A pair-only approach is clearly not always possible in this situation. In developing Chaste, regular sessions each week are devoted to developing key functionality in pairs, but development is also done individually. As part of this research project, I have been involved in these regular development sessions for the whole of the cell modelling component of the software, co-developing areas such as output, implementation of new models and refactoring code along with the rest of the team. However, the parallelisation and case study that is described in this thesis was carried out independently.

The test-driven component of agile development has been a key tool in this research project. Unit testing our additional parallel functionality gives confidence that each component is working correctly. Furthermore, since we have developed a parallel algorithm and retrofitted it into an existing software implementation of a model, where many of the key class components are shared between multiple other model implementations, great care has been taken to ensure that none of the serial functionality is broken. The unit tests provide a simple indication of any subtle changes that would not have been obvious without them.

3.3.2 Implementation of cell-based models

The main classes involved in a cell population simulation are: cells, each with a cell cycle model, a mesh, a population, a collection of population modifiers and a simulation class. The collaboration between the abstract classes in a general cell population simulation is shown schematically in Figure 3.2.

Cell cycle

The cell cycle class governs the passage of a cell object through the cell cycle phases. It holds a number of member variables for the duration of each phase of the cycle. Its main method is `ReadyToDivide()`, which is called at each time step and progresses one time increment through the cycle. The method returns a boolean value indicating whether the cell has reached M phase and should divide. This method may be altered in subclasses of the base class `AbstractCellCycleModel` to implement mechanisms such as contact inhibition or gene networks.

Cell

Each model cell is realised as a `Cell` object, which has a member variable for a cell cycle, and others that define whether it is going through apoptosis. In a software sense, a cell in Chaste stores no data on its geometry, location or neighbours. This allows different model types (such as overlapping spheres or vertex models), which define cell location and geometry differently, to share a common `Cell` class, increasing interchangeability between model types.

Mesh

Cell location, geometry and neighbourhood are encapsulated in an `AbstractMesh` class. For our overlapping spheres model, this consists of a collection of `Node` objects, which define a point in space and may store some additional data, such as an applied force. The mesh class also defines neighbourhood topology. In the overlapping spheres case, this is

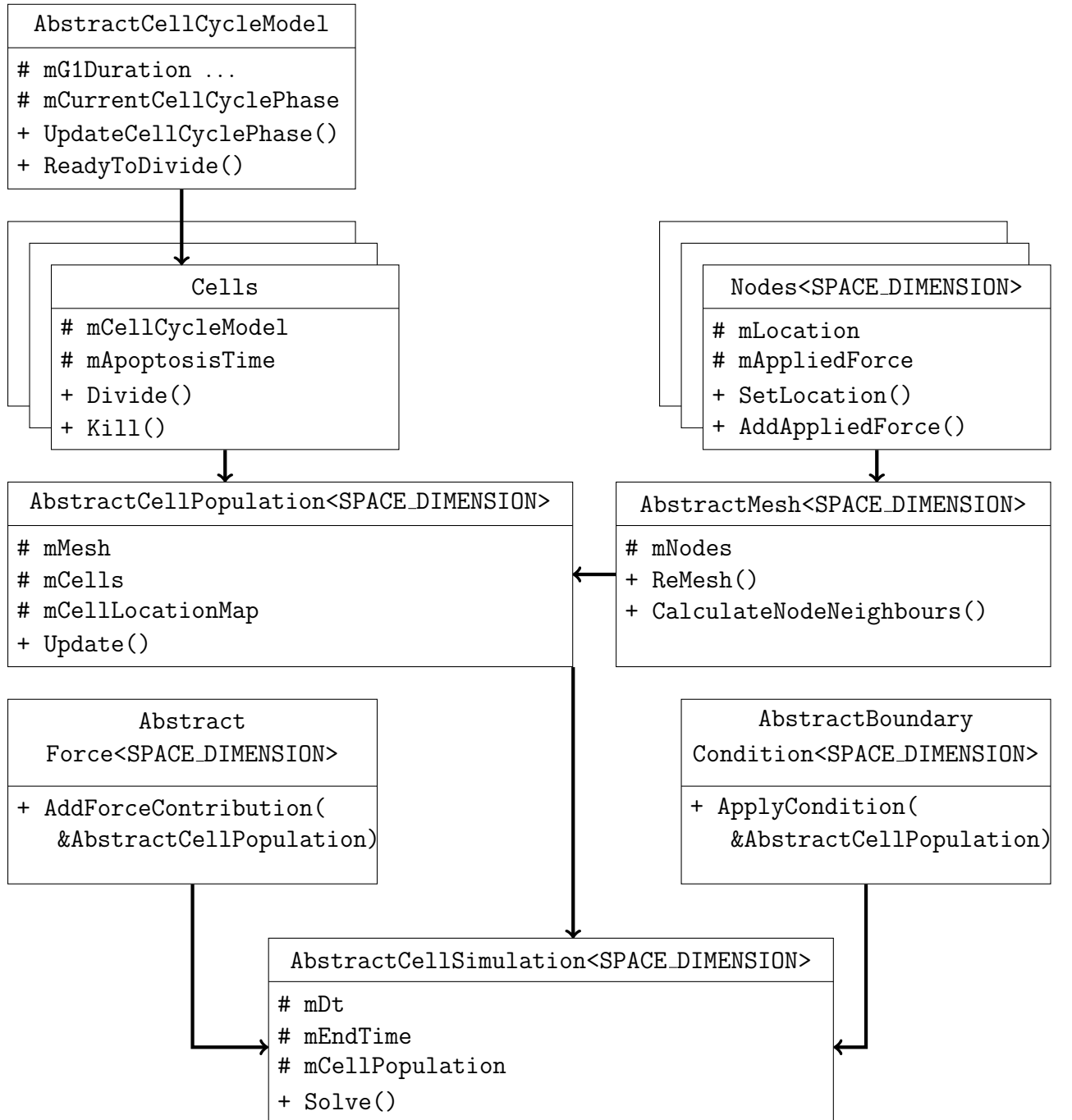


Figure 3.2: The collaboration structure of the main classes used in a cell-based simulation in Chaste. We show examples of the key member variables and functions in each class. For each class a ‘#’ represents a member variable, and a ‘+’ a member function. ‘<SPACE_DIMENSION>’ indicates that the class is templated over space dimension, so may be instantiated in either 1D, 2D or 3D.

calculated by a `CalculateNodePairs()` method, which identifies all pairs of cells satisfying the neighbourhood criterion described in Section 3.1. The `Node` and `AbstractMesh` classes are templated over space dimension, meaning that they can easily be created for 1D, 2D or 3D space using the same source code.

Population

The collection of cell objects are identified with their geometry and location in a class called `AbstractCellPopulation` by storing the cells, mesh and a mapping between them as member variables. The principal method in the cell population is `Update()`, which ensures that the mesh and cell collection are kept up to date. The population class also handles writing simulation data to file. There is a different population class for each model type implemented in Chaste, but code common to multiple model types is abstracted into parent classes, forming a hierarchy shown in Figure 3.3. By bringing common code into parent classes it is possible to minimise duplicate code and also enables faster implementation of new models.

Modifiers

A number of classes are defined that may modify the state of a cell population. For example, force classes define the forces between cells, and act through a method which is called `AddForceContribution()`, which takes a reference to a cell population object as a parameter and may add force components to the nodes associated with cells. Other modifiers include boundary conditions, which, for example, can impose a hard-wall condition along a plane, across which cells may not move.

Simulation

The population and modifier classes are combined in an `AbstractCellBasedSimulation` class, which manages the iterative progression of the model. A `Solve()` method sets up model variables and iterates through the simulation in time steps Δt , each time updating

each cell's cycle phase, orchestrating cell division, applying population modifiers and instructing the population class to write data to file. The choice of time step Δt is model dependent, and the appropriate value depends on the force laws being used. A typical approach is to choose a value of Δt such that a further halving of the time step produces quantitative results that are within some bound of the results using the larger time step. Although adaptive time stepping is not implemented in Chaste, a bound on the movement of each cell in a single time step, equal to twice its resting radius is imposed and the simulation exits with an error if this bound is broken as an indication of an inappropriately large time step.

Coupled PDE models

Although not considered for parallelisation in this thesis, in serial it is possible to solve coupled PDE models of extracellular solute transport in Chaste. The PDEs are associated with the population through the simulation class, and are solved on a separate mesh using a finite element method [Reddy, 1993]. An example application of using this functionality for modelling nutrient-limited growth of a tumour spheroid is described in Mirams et al. [2013].

Running simulations

Cell population simulations in Chaste are constructed and run as tests, similarly to the unit tests previously described. All the relevant cell cycle, cell, mesh, population, modifiers and simulation objects are constructed and the `Solve()` method called on the simulator object. These test files can then be compiled and executed. Although this requires users to understand some C++, a number of example tutorials are provided with the code, which act as a suitable starting point.

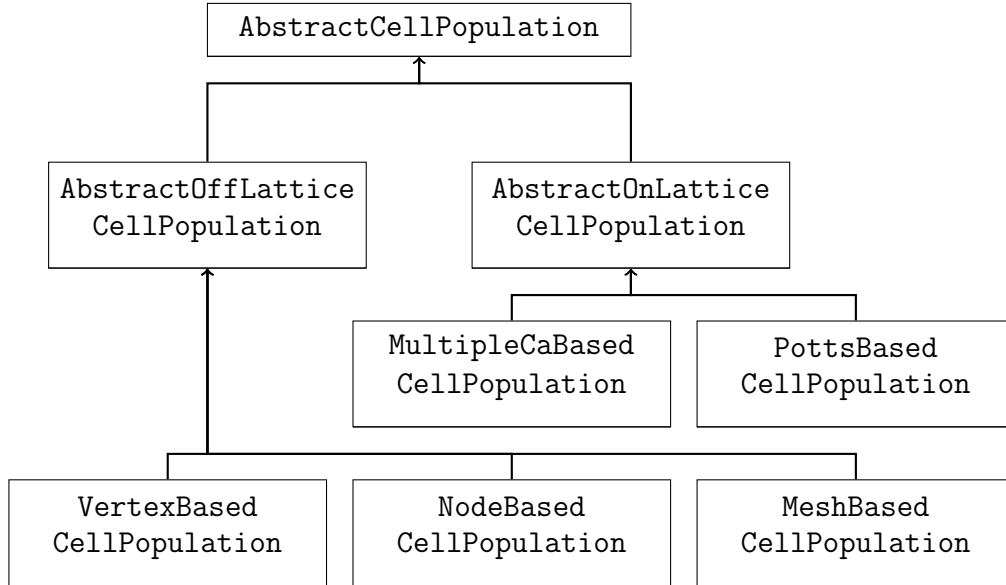


Figure 3.3: The dependency diagram for the class hierarchy of the cell population classes in Chaste.

3.3.3 Physical units

The main physical quantities in the cell modelling component of Chaste are length and time. In the code, length is in units of $10\mu\text{m}$ (approximately a cell diameter) and time in units of hours (the time scale of cell division). While this choice is suitable for modelling growth in cell populations, as the range of applications grows, care must be taken to ensure consistent units are used. This is a pervasive problem in scientific computing, where a floating-point value could represent any type of unit, and, in a general programming language such as C++, it is not simple to check that operations between variables produce values with the correct units. As described by Stroustrup [2012], the new C++11 specification [C++11, 2011] provides the opportunity for defining the SI system as suffix operators, so that a developer may write, for example

```
Length distance = 10m;
```

to specify that `distance` is a length, and is measured in meters. The consistency of units may then be checked at compile time. As the C++11 standard becomes more widely available in compilers, this may provide a solution to the problem of specifying units.

3.3.4 Simulations in Chaste

As an example of the capabilities of the software implementation of this model, we describe an example simulation published recently by Mirams et al. [2013]. A snapshot of this simulation is shown in Figure 3.4. The model represents cell division, signalling, differentiation and migration in a small patch of the lining of the murine small intestine that consists of a single villus projection surrounded by four crypt invaginations. The geometry is imposed using a boundary condition that restricts cells to move on the surface of the crypts and villi. Proliferative stem cells located at the base of the crypts divide, generating a force which drives cells along the crypt-villus axis and populates the patch of tissue.

Cell-cell mechanical interactions are governed by the force law introduced in Section 3.1, with cell proliferation in the base of the crypt driving cell migration. The progression of cells through the cell cycle phases is governed through two external factors. A linear gradient of Wnt signalling is imposed along the crypt-villus axis, in line with the model used by van Leeuwen et al. [2009], which gives cells a cue to change into a differentiated state, where cell division no longer occurs, as they migrate up the axis. Each cell is also endowed with an ODE model of Delta-Notch lateral inhibition, first proposed by Collier et al. [1996]. The ODE depends on the Delta levels in neighbouring cells, which are also governed by the same model, creating a feedback mechanism which leads to a patterning of high and low Delta levels. This is believed to consign cells to a secretory or absorptive fate giving rise to the pattern of these cell types seen *in vivo*. This pattern can be seen in the right hand image in Figure 3.4. Furthermore, each cell is labelled with an identifier for its parent cell, allowing the clones of different cells to be tracked throughout the simulation. The progeny of a single stem cell are seen to populate an entire crypt as a result of mechanical competition, and consequently the villus eventually becomes populated by the progeny of four cells, one from each surrounding crypt (Figure 3.4, left-hand image).

As we noted in the previous chapter, Chaste has functionality to simulate a range of

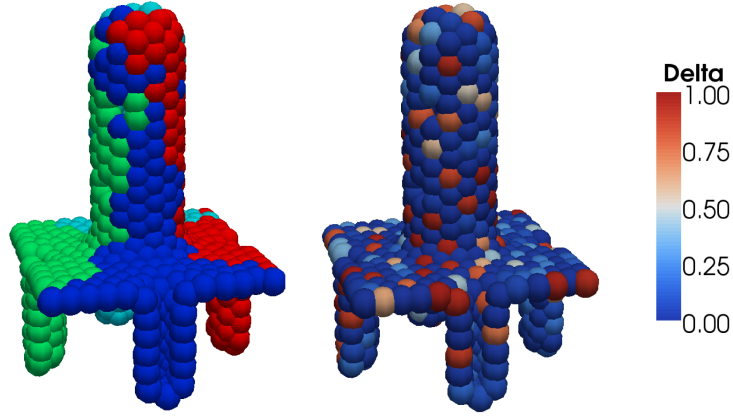


Figure 3.4: An example simulation using the model described in Section 3.1. The division, migration, signalling and differentiation of cells in a small patch of murine small intestine is simulated using the Chaste software library. Left hand image: the progeny of a single cell from each of the four crypts colonise the entire patch of tissue. Right hand image: Lateral inhibition in an internal model of Delta-Notch signalling in each cell leads to a pattern of different cell differentiation types. Reproduced with permission from Mirams et al. [2013].

other cell population models that have been developed in the literature. In addition to the overlapping spheres model described in Section 3.1 and shown in Figure 3.5(a), it is also possible to simulate the Voronoi-Delaunay cell centre model as shown in Figure 3.5(b), which has previously been used to develop a model of epithelial growth and the geometry of the colorectal crypt [Dunn et al., 2012a,b]. There is also functionality to simulate vertex dynamics models shown in Figure 3.5(c), which have been compared with the cell centre approach in a model of colorectal crypt dynamics [Osborne et al., 2010]. The cellular Potts model shown in Figure 3.5(d) has been implemented within Chaste, and will be the subject of a future publication. Each model implementation differs in its geometry and definition of cell neighbourhood, but share a common approach to simulating cellular processes, such as the cell cycle.

In order to provide a comparison of their computational cost, as implemented in Chaste, we simulate 10 hours of a basic example of each model where a 2D square patch of cells is created with a simple fixed duration cell cycle model. We choose 2D to allow comparison with the vertex dynamics model, which is currently only available in 2D. No division occurs during the simulation. The observed wall time for each simulation is

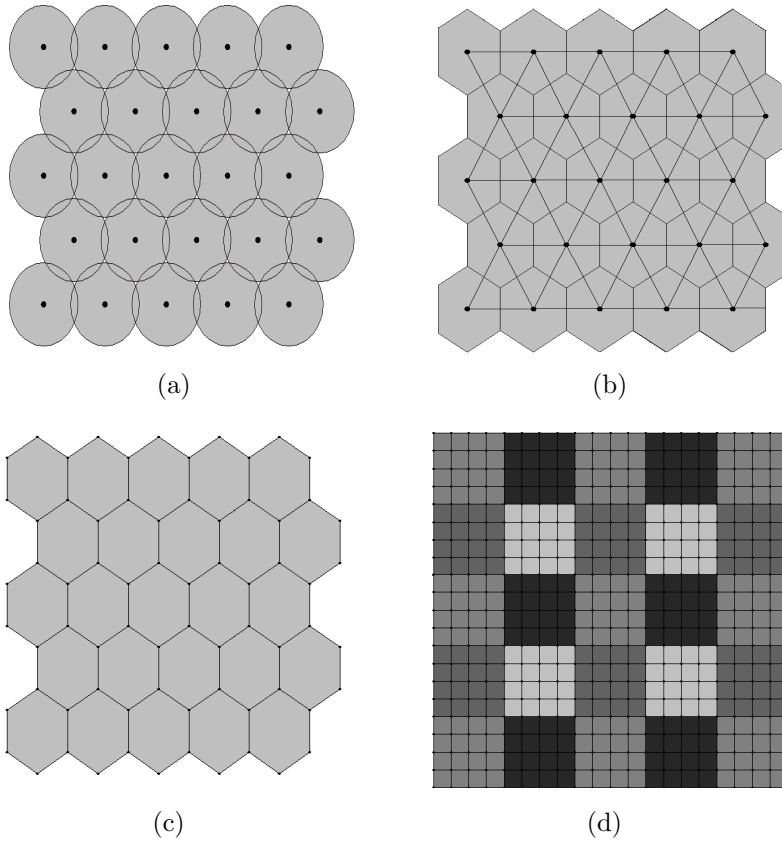


Figure 3.5: Visualisation of the four principal cell population modelling approaches available in Chaste. (a) The overlapping spheres (circles) approach described in this section. (b) The Voronoi-Delaunay approach. (c) The vertex dynamics model. (d) The cellular Potts model where each cell initially occupies 16 lattice sites.

plotted on a logarithmic scale in Figure 3.6(a), where it can be seen that the wall time for simulating each model scales approximately linearly with the number of cells and there is no significant difference between the models. However, it is noticeable that the cellular Potts model is the most computationally costly in this comparison. This is because each cell is composed of 16 lattice sites, which significantly increases the computational cost compared to the cell centre and vertex dynamics models which have fewer degrees of freedom per cell.

For the overlapping spheres model which we consider in this chapter, we plot the proportion of computation time spent in the most costly operations in Figure 3.6(b). These three components are: ‘Update’ where the cell cycle model of each cell is progressed, and the neighbourhood topology is updated by calculating the nearest neighbours of each cell; ‘Force’ which calculates the cell-cell interaction forces based on the force law described in Section 3.1; and ‘Position’ which carries out integration of the equation of motion for each cell using the forward-Euler method in equation (3.2). It can be seen that ‘Update’ and ‘Force’ dominate the cost of simulating this model, and in particular the cost of updating the neighbourhood of each cell in ‘Update’ scales poorly as the number of cells simulated is increased. The aim of the next two chapters will be to develop an efficient parallel algorithm that will reduce the cost of these two components, enabling larger and faster simulations to be carried out.

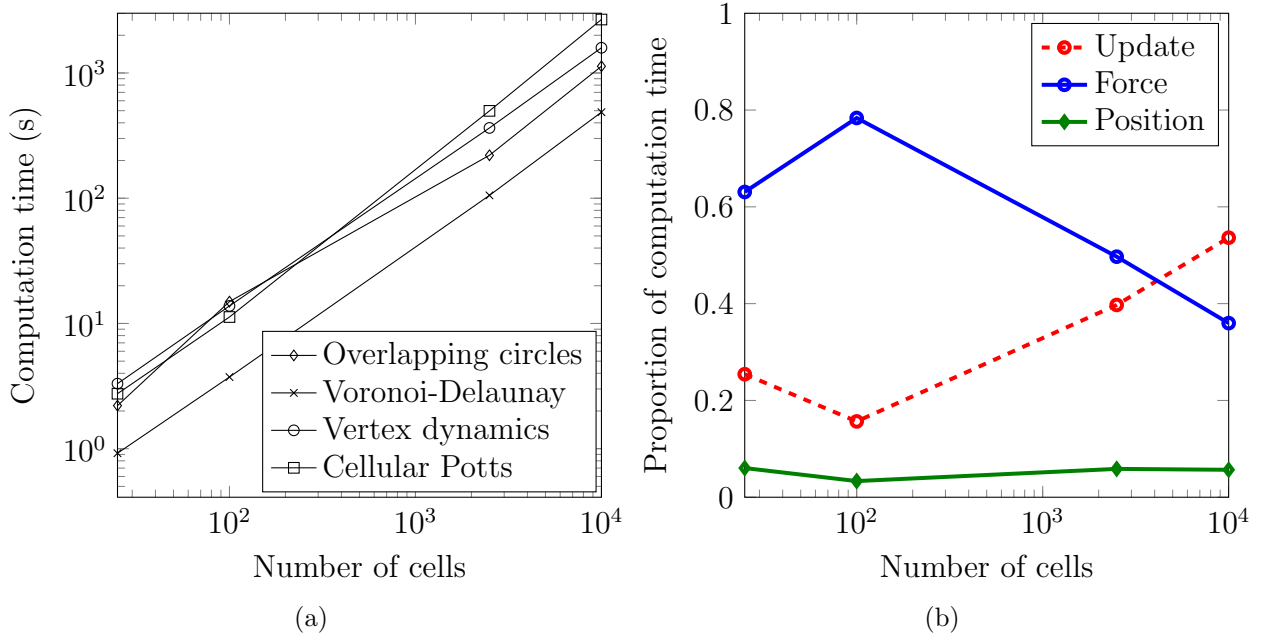


Figure 3.6: (a) A comparison of the computation for a simple 2D model using each of the modelling paradigms available in Chaste as cell numbers are increased. (b) For the overlapping spheres (circles) model, the main components of the computational cost of the algorithm. ‘Update’ includes updating the cell cycle model of each cell and re-generating the cell nearest-neighbour topology. ‘Force’ includes the calculation of cell-cell mechanical forces using the force described in Section 3.1. ‘Position’ includes updating the location of each cell in accordance with the over-damped equation of motion.

Chapter 4

Parallelising an off-lattice model of a cell population

Computational cost places substantial restrictions on novel research in computational biology and physiology. In this chapter we describe a parallel algorithm for off-lattice centre-based models of populations of cells that will reduce computational cost and accelerate novel research in this field. The algorithm divides the spatial domain between processors, and comprises communication routines which ensure the model is correctly simulated on multiple processors. The algorithm is implemented in Chaste, and is shown to accurately simulate the model described in Chapter 3. We also show that, while we achieve speed-up, the algorithm scales relatively poorly due to high levels of inter-processor communication and propose strategies for reducing this inefficiency.

4.1 Introduction

As a consequence of the cost of calculating pairwise cell-cell interactions when simulating the model described in Chapter 3, its applications are limited by current computer resources to $10^4 - 10^5$ cells, and simulations of this size take a prohibitive amount of time to compute. Increasing this limit will broaden the scope of applications of the model.

Taking chemotherapeutic strategies for treating cancer as an example application, the current scale of available simulation ($10^4 - 10^5$ cells) corresponds to the early avascular stage of tumour growth [Galle et al., 2005, Drasdo and Höhme, 2005], while a tumour is not typically detectable by X-ray until it has grown much larger, to approximately 10^8 cells [Alberts et al., 2002]. While simulations with 10^8 cells may be beyond the scope of even a distributed algorithm on a large HPC cluster, further development of novel therapies using computational modelling approaches depends on increasing the total number of cells that we can simulate.

The computational intensity of the model we consider gives rise to two related problems. First, as stated, there are scales of multi-cellular systems that we cannot currently simulate. Second, running simulations at the upper limit of current tractability is extremely time-intensive. The former limits the reach of the model, but the latter also restricts how it can be used, for example, by preventing a systematic parameter sensitivity analysis [Byrne and Drasdo, 2009]. These two problems are related since increasing the speed of simulation of tractable models, whilst keeping the available computational resources constant, will inevitably make it possible to simulate larger models. However, addressing the speed of simulation will not suffice: the scale of simulations is limited not only by processing capacity, but also by the amount of memory available on the computer. Distributed-memory parallel computers offer substantial increases in available memory, but necessitate a parallel algorithm.

In this chapter we attempt to address these problems by presenting a distributed algorithm that can be used to simulate our model on distributed-memory HPC technology. These machines contain multiple processors, which are connected by a communication network that allows each processor to copy a portion of its local memory to the local memory of another processor in the machine. Distributed-memory machines offer increased processing and memory capacity beyond that of other HPC architectures, and consequently provide us with the possibility of simulating much larger models.

The development of a parallel algorithm for this model primarily involves dealing with

two significant practical issues: how best to decompose the data so that the computational task is spread between the processors; and how best to communicate data between processors so that the algorithm accurately simulates the model. To address these issues, we choose to decompose the data by assigning to each processor a fixed portion of the spatial domain in the model, and we develop a novel approach that uses functions primarily designed for saving and loading the state of simulations to communicate data between processors.

The principal result of this chapter is to demonstrate that our algorithm correctly simulates the model. This is achieved by showing that results of simulating a deterministic version of the model are independent of whether the serial or parallel algorithm is used. As a precursor to the work on computational optimisations in Chapter 5, we also test the efficiency of our algorithm by simulating five different cell populations, each with a different spatial distribution of cells. By considering the parallel speed-up achieved along with the computational cost of different parts of the algorithm, we show that the level of detail associated with each cell, as well as load-balance problems, limit the scalability of our basic algorithm. We define the speed-up for k processors as the ratio of the computation time for the serial algorithm to the computation time for the distributed algorithm using k processors.

In the next section we review the approaches that have been taken when using HPC for other types of cell population models, and discuss relevant approaches in the related MD field. In Section 4.3 we describe our distributed algorithm, and outline the key features of its implementation in Chaste. We present the results of validating our algorithm and testing its scalability in Section 4.4, and discuss the significance and future development of this work in Section 4.5.

4.2 Previous work

Calculating cell-cell interactions (both the neighbourhood topology and interaction forces) consumes the most CPU time when simulating the model described in Chapter 3. Here, we review parallel approaches that have previously been used to overcome this problem for both individual-based cell population models, and the related discipline of MD. Since they typically execute calculation of short-ranged interactions between particles, we describe parallel approaches that have been developed for MD simulations. A detailed description of MD lies outside the scope of this work; hence we refer the interested reader to the recent book by Marx and Hutter [2012] and focus here on aspects relevant to our parallel algorithm. In Section 4.2.2 we review parallel algorithms used in the context of cell population simulations. The widespread popularity of the Graner-Glazer-Hogeweg (cellular Potts) model described in Chapter 2 has made it the prime candidate among individual-based models of cell populations for the development of parallel algorithms [Chen et al., 2007]. However, more detailed models, such as the ScEM [Newman, 2005], are the subject of recent developments of parallel algorithms for GPU systems [Christley et al., 2010].

4.2.1 Parallel approaches in molecular dynamics

At their core, MD simulations evolve the state of particles according to a predefined model. In particular, the spatial location of each particle is typically updated by integrating Newton’s equations of motion for the balance of forces on the particle. Since these forces often arise from interactions with nearby particles, the cost of calculating particle neighbours and particle-particle forces arises in MD as it does in cell population models. Since a large body of work has been devoted to solving this problem in MD, we review both serial algorithm optimisations and methods of parallelisation to inform the development of our own method.

4.2.1.1 Comparing MD and individual-based cell population models

To calculate pairwise interactions in many MD simulations it is necessary to identify particles which are separated by a distance less than some fixed constant distance, r_c . We shall henceforth call a pair of particles that satisfy this condition neighbouring particles. An analogous problem arises in simulating the cell population model described in Chapter 3 where, to calculate cell-cell interactions, we wish to find all pairs of cells whose centres are separated by less than some fixed distance. In both cases the computational cost of calculating the set of neighbouring particles at each time step dominates the overall cost of the simulation when the number of particles or cells is large.

There are, of course, many conceptual differences between MD and cell population models, which lead to differing characteristics of the simulation algorithms. In typical MD simulations the particles have no physical extent, or at most a simple model of constant radius, whereas in the cell population model we consider each cell to have a non-zero volume defined by a sphere around its centre. If the radius of the sphere is a state variable of a cell, which may change over time in response to extracellular signals, then our neighbourhood criterion depends on the radii of the two cells being considered. If two cells i and j at locations $\mathbf{x}_i$ and $\mathbf{x}_j$ have radii R_i and R_j respectively, then they are neighbours if and only if $\|\mathbf{x}_i - \mathbf{x}_j\| < R_i + R_j + r_{\text{int}}$, where $r_{\text{int}} > 0$ is a fixed constant. This difference is not of great significance, however, since in the cell population model the radius of each cell is bounded above: once a cell has grown to double its natural radius, R , it divides into two daughter cells with radii R . Therefore all potential neighbours can be identified from the MD techniques outlined above by setting $r_c = 4R + r_{\text{int}}$.

There are further fundamental differences between the level of detail considered for each individual in MD and cell population models. In typical MD simulations the location of each particle is stored, possibly along with a few state variables representing, for example, the type of particle. In contrast, in a simulation of the cell population model we must store data for each cell on its type, its progress through the cell cycle, and where appropriate the state of subcellular ODE systems. This characteristic difference is likely to affect the choice

of parallel decomposition, particularly when a distributed-memory architecture is used. A more detailed description of each individual leads to a greater cost of communicating the data associated with an individual between processors. Therefore, it is expedient to choose a decomposition that minimises the total amount of communication. In addition, when the parallel decomposition requires data on every individual in the model to be stored in each processor’s memory, a detailed model is likely to be highly restricted in the number of individuals that can be simulated. This is because each processor has access to a limited amount of memory, which will be exhausted if a large number of detailed individuals are considered.

A further significant difference between the models is that some cell population simulations exhibit a large increase in cell numbers over time due to proliferation, whereas MD simulations typically comprise a fixed number of molecules. This could cause a load-balancing problem since cell numbers are not known *a priori*, leading to a requirement for dynamic load-balancing in simulations of growing cell populations.

While we have outlined some of the sources of increased computational intensity of cell population simulations compared to their MD counterparts, there are a number of challenges faced in MD that do not arise in simulating cell population models. The most significant of these is that MD models typically include long- as well as short-ranged interactions between particles, requiring the development of novel simulation schemes for long-ranged interactions, such as Ewald summation [Darden et al., 1993].

4.2.1.2 Efficient serial algorithm for neighbour calculation

The naïve approach to calculating the neighbour list is to loop over each possible pair of particles and evaluate the distance between them, storing the pair only if the neighbourhood criterion is satisfied. This calculation has $O(n^2)$ complexity, where n is the number of particles in the simulation, and is therefore likely to dominate the computation time as larger systems of particles are considered. However, the short-ranged nature of the neighbourhood criterion may be used to reduce this computational cost.

In the defining work on this topic, Plimpton [1995] describes an optimisation to the neighbour calculation algorithm, originally due to Verlet [1967]. When the neighbour list is constructed a larger cut-off $r_s = r_c + \delta$ is used. Only particles within a distance r_c are used to calculate particle interactions, but since a list from an increased neighbourhood is stored, it only needs to be updated every few time steps, or (if limits on the velocity of particles are known *a priori*) when the maximum possible distance moved by any individual particle means that the neighbour list is incorrect. This method is particularly effective when the distance moved by particles over each time step is small compared to the cut-off distance, since this type of simulation will not require regular re-calculations of the full neighbour lists.

A further optimisation may be achieved through the cell-list method [Fincham, 1987]: provided the size of the physical domain is large compared to the force cut-off distance r_c , we can coarsely divide the domain up into cubes of dimension r_c (or larger). Each particle then lies in a unique cube, and it must be the case that all of its neighbours lie in the same cube or surrounding cubes and so only particles within these cubes need to be tested. This substantially reduces the number of pairs that must be considered; indeed, supposing that the domain can be split into m boxes with each containing an equal number of particles, we must only search over $n(27n/m - 1)$ possible pairs. For large domain sizes this can be a substantial optimisation. It is actually possible to do better: since the neighbourhood criterion is symmetric, we only need to consider half of the surrounding boxes, so the search may be reduced to considering $n(14n/m - 1)$ possible particle pairs. These serial optimisations can reduce the computation time in serial and we have used the cell-list method for our simulations of the cell population model in Chaste. However, to achieve large-scale simulations it is necessary to make use of the increased processing power and available memory of parallel HPCs.

4.2.1.3 Parallel decomposition approaches in MD

In this section we describe approaches for simulating MD models on HPCs because of the fundamental similarities between these algorithms and our model, and because much more development of parallel algorithms has occurred in the MD field. We focus on parallelisations made for distributed- and shared-memory multiprocessor computers, since this is by far the most popular approach to date for simulating MD models. We also discuss the growing trend for using GPGPUs, as well as hybrid systems, to aid the simulation of MD models. For each method we focus on how the data (particles) are decomposed between processors in a parallel environment, and the consequences for particle interaction calculations, since these aspects are most relevant to the development of a parallel algorithm for our cell population model.

Neighbour calculation is a problem that lends itself well to parallelisation since it is a sequence of independent calculations. The configuration of the particles is updated synchronously once all pairs have been identified and therefore identifying one pair does not affect the data relevant to other neighbour calculations. Three principal approaches have been used to calculate neighbours in parallel MD simulations [Plimpton, 1995]. These approaches are termed Atom Decomposition (AD), Force Decomposition (FD) and Space Decomposition (SD). Although developed over 20 years ago, these remain the primary methods for calculating pairwise neighbours in parallel MD simulations [Afshar et al., 2013].

In order to describe the AD and FD methods it is useful to define the neighbourhood matrix $F = (f_{i,j})$, where $f_{i,j} = 1$ if cells i and j are neighbours and $f_{i,j} = 0$ otherwise. For large cell populations F will be sparse and symmetric since neighbourhood is a commutative property.

Atom decomposition: The AD method assigns each particle a unique processor by striping the neighbourhood matrix along its rows, as shown in Figure 4.1(a). Suppose, for simplicity, that the number of processors k divides the number of particles n . Then

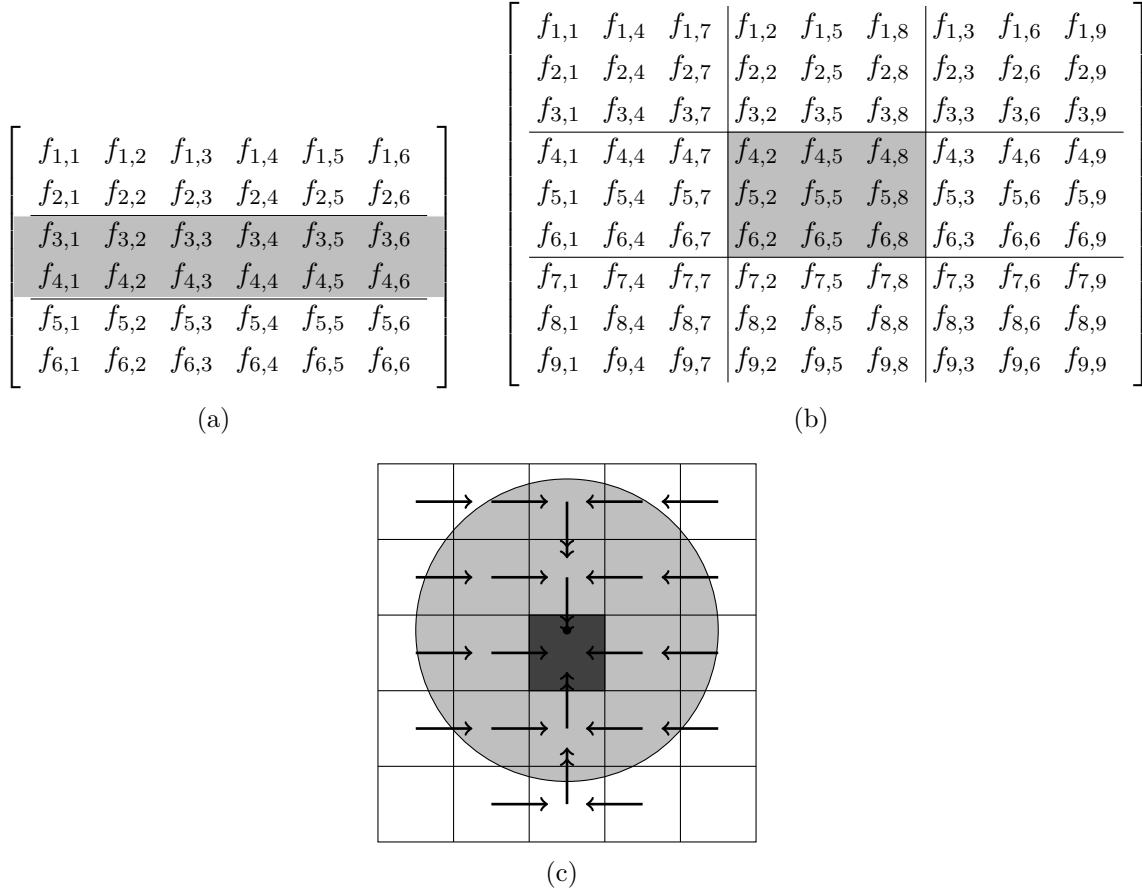


Figure 4.1: Data decomposition methods. (a) The AD method. The rows are divided among the processors, with the calculation of each block of rows assigned to a processor. Each processor requires information on the location of every particle in the simulation, which results in a communication overhead that scales with n , the number of particles. (b) The FD method. The matrix is blocked, and the responsibility for calculating each block assigned to a processor. In this approach the amount of communication is reduced compared to the AD method, so that it scales with $n/\sqrt{k}$, where n is the number of particles and k is the number of processors. The shaded region shows the interactions to be calculated by one processor. (c) The SD method, where the physical domain is divided into cuboids with each cuboid assigned to a processor. For clarity we show the 2D case here, which extends in a straightforward manner to 3D. Communication between neighbouring boxes takes place along a co-ordinate axis direction until all cuboids know the location of particles within the cut-off distance of their boundary. This process is repeated in the remaining co-ordinate directions until each processor has all particle locations needed to calculate the neighbours of particles that it owns. The communications scales approximately as nr_s/L where n is the number of particles, L is the linear size of the space domain and r_s is the neighbour cut-off distance.

processor 0 is assigned the first n/k particles, processor 1 the next n/k particles, and so on until all particles are assigned (if k does not divide n , then the last processor is assigned fewer particles). Each processor is responsible for calculating the neighbours and updating the locations of the particles that it owns. To do so, each processor must know the location of every particle in the entire simulation at each time step, and this information must therefore be communicated between processors. Berendsen et al. [1995] use this decomposition approach in their widely used parallel MD simulation software package GROMACS.

One of the benefits of the AD method is that it is simple to implement, compared with other decomposition approaches. Few changes are required to the serial algorithm code beyond setting up and synchronising the appropriate communication between processors. Furthermore, AD does well at balancing the computational load between each of the processors. Since each processor owns the same (or nearly the same, if k does not divide n) number of particles, an equal number of possible neighbours are tested on each processor.

While the AD method is well suited to simulations of modest numbers of simple particles, the all-to-all communication step is a significant drawback for large or more detailed models [Plimpton, 1995]. In this case the cost of the latency of the communication becomes so large that it dominates the computation time. A further inefficiency of the AD approach is the difficulty in exploiting the fact that F is symmetric. Since $f_{i,j}$ and $f_{j,i}$ will rarely be evaluated by the same processor, to apply the symmetry property the calculated value of $f_{i,j}$ must be communicated to the processor owning particle j , or *vice versa*: in effect the computation cost is halved but the volume of communication is doubled.

Variations on the AD scheme have been proposed to address these problems. In simulating polymer chains of particles, Jung et al. [2000] avoid large all-to-all communications by first executing a coarse-grained neighbour calculation. Each chain of particles is owned by a unique processor. Axis-aligned bounding boxes are created around each chain of particles, so that each particle chain is contained within the union of the boxes for that chain. Possible neighbours with particles in other chains on other processors are discounted when

the bounding boxes for the two chains are separated by a distance greater than r_c . In addition, when boxes for a pair of processors are found to be separated by more than r_c , separating co-ordinate planes between the boxes are stored. Only when a particle from one of the chains crosses a plane must the coarse-graining step be repeated, otherwise the boxes remain disjoint. Although this approach is reported to achieve good levels of speed-up for this specific type of simulation, its application to a wider class of MD simulations would lead to significant complications in implementation. In particular, how to choose the size of the coarse-grain boxes is highly problem-specific, and dependent on the spatial distribution of particles on each processor. To benefit from the coarse-grain step the boxes must be large enough that they contain more than a few particles. On the other hand, if the boxes are too large few of them will be disjoint and the benefits of the method are again lost. Due to these complications this method has been overlooked in favour of the simpler AD method, or more efficient FD and SD methods described below.

The AD method provides a simple parallel structure for calculating particle neighbours that is naturally load-balanced. However, the high volume of communication means that speed-up is likely to be limited for large or detailed problems. The FD approach builds on the AD method by reducing this volume of communication.

Force decomposition: The FD method, first reported in Plimpton and Hendrickson [1994], improves the AD method by assigning to each processor a subset of possible neighbours to evaluate. This corresponds to blocking a column-permuted version of the neighbourhood matrix F and assigning each block to a processor. Figure 4.1(b) shows how this is achieved in the simple case where $n = 9$ and $k = 9$. The columns of F are re-ordered as $[1, 1+n/\sqrt{k}, 1+2n/\sqrt{k}, \dots, 2, 2+n/\sqrt{k}, 2+2n/\sqrt{k}, \dots, \dots, n/\sqrt{k}, 2n/\sqrt{k}, \dots, n]$. The shaded block in Figure 4.1(b) shows the interactions to be evaluated by one processor. For this method each processor only needs to store location data for $n/\sqrt{k}$ particles. The amount of communication for this method therefore scales with $n/\sqrt{k}$, compared to scaling with n for the AD method. The FD method also retains the load-balancing benefit

of AD.

This procedure is a substantial improvement over the AD method, as the relative communication overhead is reduced as more processors are used, leading to much greater parallel efficiency, achieving close to perfect linear speed-up for up to 1,024 processors for a simple MD simulation of 14,026 particles [Plimpton and Hendrickson, 1994]. However, the FD method fails to exploit the short-range property of the neighbour criterion. This is a significant drawback when communication costs are high, and we wish to avoid checking pairs that are separated by a large distance and are owned by different processors.

Space decomposition: The third major parallel procedure for calculating neighbouring particles is the SD method, in which the physical domain is divided into spatial regions and each processor is assigned one or more of these regions. A processor owns all particles contained within its regions. The typical approach, and that described by Plimpton [1995] is to use cuboidal regions which simplifies the definition of neighbouring regions and the computational implementation of the method. Therefore, we restrict our description of the SD method to cuboidal regions.

To calculate the neighbours of a given particle contained within a given cuboid, the particle's owning process must know the location of all other particles that lie in adjacent cuboids. Suppose that each cuboid has dimension d_l for $l = 1, 2, 3$ for the x , y and z co-ordinates respectively. The first step is for possible neighbours to be calculated in the east-west direction (parallel to the x -axis). The particles that are within a distance r_s (the enlarged cut-off described in Section 4.2.1.2) of the cuboid to their east are sent to that cuboid, and corresponding data received from the west. If $d_l < r_s$ this constitutes all the particles in the adjacent cuboids, otherwise the process is repeated with cuboids a step farther apart until all particles within a distance r_s of the cuboid have been received from the west. The process is then repeated for sending to the west and receiving from the east. The location data of particles received from neighbouring processors (so-called halo particles) are stored in the memory of the processor along with the data on the particles

that it owns. This step is then repeated in the north-south direction, except that *all* data, including halo particles, are sent in each direction. Finally the process is repeated in the third spatial dimension. Figure 4.1(c) shows a 2D schematic for this operation. At the end of these steps each processor has data on all particles that lie within a distance r_s of its cuboid. The neighbours are then evaluated and the forces calculated to enable the particle locations to be updated. If a particle moves out of the cuboid for its processor, then it is transferred to the appropriate processor by a communication, and deleted from the memory of the processor that previously owned it. This parallel algorithm is an augmented version of the cell-list method, where the cells (cuboids) are allowed to be smaller than the neighbourhood cut-off. It also uses the extended cut-off radius r_s so that the full neighbour calculation does not need to be used at every time step.

The primary benefit of the SD method is that communication occurs between local sub-sets of the processors, rather than all processors. This significantly reduces the cost of communication, without increasing the cost of computing the neighbours on each processor. Furthermore this method can be significantly more memory efficient than the AD and FD methods. Since each processor only stores approximately n/k particles, and as few as $26n/k$ halo particles, much larger populations of particles may be simulated for a fixed amount of memory per processor. Plimpton [1995] reports substantially greater speed-ups for the SD procedure; indeed, the author found that it was possible to run simulations of two orders of magnitude more particles with the SD method than using the AD or FD methods for 1,904 processors.

However, the SD method does not represent a panacea for particle neighbour calculation. Unlike the AD and FD methods it suffers from being particularly difficult to load-balance. To achieve perfect load-balance, particles would have to be uniformly distributed throughout the domain at all time steps, otherwise some processors will have to communicate more halo particles and calculate more pairs than others. This presents a substantial challenge for the implementation of this algorithm since the optimal load-balance will be dependent on the distribution of particles in a simulation. The aim, in

order to achieve optimal speed-up, would be to have an equal number of particles being owned by each processor.

We have briefly reviewed the three main approaches to developing a parallel algorithm for calculating neighbouring particles in MD simulations on HPCs. The literature suggests that there is a trade-off between the simplicity with which a serial algorithm can be adapted to parallel and the resulting efficiency of the parallelisation. The AD method offers the simplest approach, yet suffers from high volumes of communication which limits the size and detail of the models for which it can be used. The FD method reduces the communication burden by decomposing the pairs to be evaluated, rather than individual particles. Although the amount of communication is reduced, much of it is redundant, since no spatial information on the location of particles is considered. Both methods also suffer from high memory consumption on each processor, which would be exacerbated with a model of a cell population, where much more detail is stored with each individual. These communication and memory problems are reduced in the SD method, which, for idealised problems, can achieve much greater levels of parallel efficiency. The SD method, however, typically suffers from poor load-balancing, because it is difficult to ensure both that each processor owns an equal number of particles and also that communication between neighbouring processors is minimised.

General purpose graphics processing units: In the last decade the increase in availability of high-performance GPGPUs, and the ability to program them for functions other than rendering graphics, has lead to their use in the MD field [Stone et al., 2007, Rodrigues et al., 2008, Liu et al., 2008, Xu et al., 2010]. GPGPUs provide extra processing power alongside the CPU, and have their own dedicated memory. The CPU and GPGPUs can exchange messages to copy sections of their memory to one another. This is typically used in MD simulations by the CPU sending packets of location data for particles to the GPGPU to calculate the neighbours, using the cell-list method. By using GPGPUs, such as the NVIDIA GeForce 8800 GTX¹, speed-ups of around 20 have been achieved for

¹<http://www.nvidia.co.uk/>

moderate numbers of particles ($10^3 - 10^4$).

Although this presents an attractive option for optimising the speed of MD simulations on conventional desktop computers, the size of problem that remains computationally tractable is unchanged. Since no additional memory is added to the CPU, and the number of additional processors is limited to those on the GPGPU, the upper limit of the size of the system that can be simulated is only marginally increased. The development of cutting-edge hybrid CPU/GPGPU machines, whose compute nodes contain both many-processor CPUs and GPGPUs, may well provide the next step in the development of parallel approaches to MD simulation in the future [Plimpton and Thompson, 2012].

In summary, the methods presented in this section do not provide an ‘off-the-shelf’ solution to developing a parallel algorithm to simulate the model of a cell population described in Chapter 3. The increased data associated with an individual in our model makes the task of communication more difficult, since we cannot simply send a vector containing the location of the cell: interactions in the cell population model can depend on any of the other properties of the cell, such as the cell type or radius. Cell division can complicate the definition of which processors own which individual and make load-balancing more difficult. In MD simulations, in contrast, there is typically conservation of the number of particles over the course of the simulation. However, we use the parallel methods for MD simulation to guide the development of our own algorithm.

4.2.2 Parallel implementation of individual-based models

Individual-based models of cell populations in general appear to offer opportunities for parallelism: they comprise a large number of separate individuals whose states must be updated according to a set of pre-defined rules. However, the problem of simulating these models on HPC technology is not an ‘embarrassingly parallel’ task (i.e. a task with little or no data dependence that is therefore easy to parallelise). The key feature of individual-based modelling is the ability to uncover emergent properties from the *interaction* of a large population of heterogeneous individuals with each other and their environment. Due

to this dependence of each individual on its surroundings, the task of parallel simulation is made non-trivial. Nevertheless, using the computational power of HPC technology has substantial benefits for scientific research. Studies that were previously computationally intractable, such as the detailed simulation of solid tumours, are potentially within reach. Simulations that take days and weeks to complete on serial architectures become realistic tools for estimating parameters by running multiple simulations. We would therefore expect to see the use of HPC technology applied to simulating individual-based models of cell populations as commonplace in the field. Bader and Aluru [2008] note, however, that the development of parallel approaches within computational biology has failed to keep pace with the advances of the rest of the field. We now review progress that has been made in this field, and the approaches that have been taken.

Effort to develop parallel algorithms for individual-based models of cell populations has largely been focused on the CPM and for the state machine agent-based models described in Chapter 2. Chen et al. [2007] report a parallel algorithm for the CPM that has been shown to be able to simulate more than 10^6 cells on approximately 25 processors. Their method decomposes a 2D lattice into a checkerboard as shown in Figure 4.2, with each subgrid containing a relatively small number of lattice sites compared to the overall total. Each processor controls the update of a square of four subgrids. This decomposition is driven by the primary problem associated with parallelising the CPM update algorithm: that changing the properties of one lattice site has non-local effects. The effect of one site is typically restricted to some radius of influence around the physical location of the site, and so when a site is updated, all sites within the radius of influence must have up-to-date information on this change before changes to their state can be considered. This condition of up-to-date information cannot be guaranteed if two sites within the interaction distance lie on different processors. To avoid this leading to inaccurate simulation of the model, the Chen algorithm updates the sites on each of the four local subgrids in sequence so that all neighbouring squares are locked while the update is carried out. Once each subgrid is updated, each processor exchanges information with its neighbours from the lattice sites

near to processor boundaries (labelled buffers), so that up-to-date data is always used to evolve the model. One might expect that imposing a wait on each processor during each update step would lead to large inefficiencies of the algorithm, but efficiencies of between 80% and 90% are reported for simulations of approximately 10^6 cells. Similar approaches to parallelising the algorithms of related lattice-based models have also been based on the SD approach, such as the cellular automaton model of tissue growth by Youssef et al. [2007], or large-scale models of the human inflammatory response by Alberts et al. [2011], but they have not reported success in the aim of simulating numbers of agents on the same scale as Chen et al. [2007].

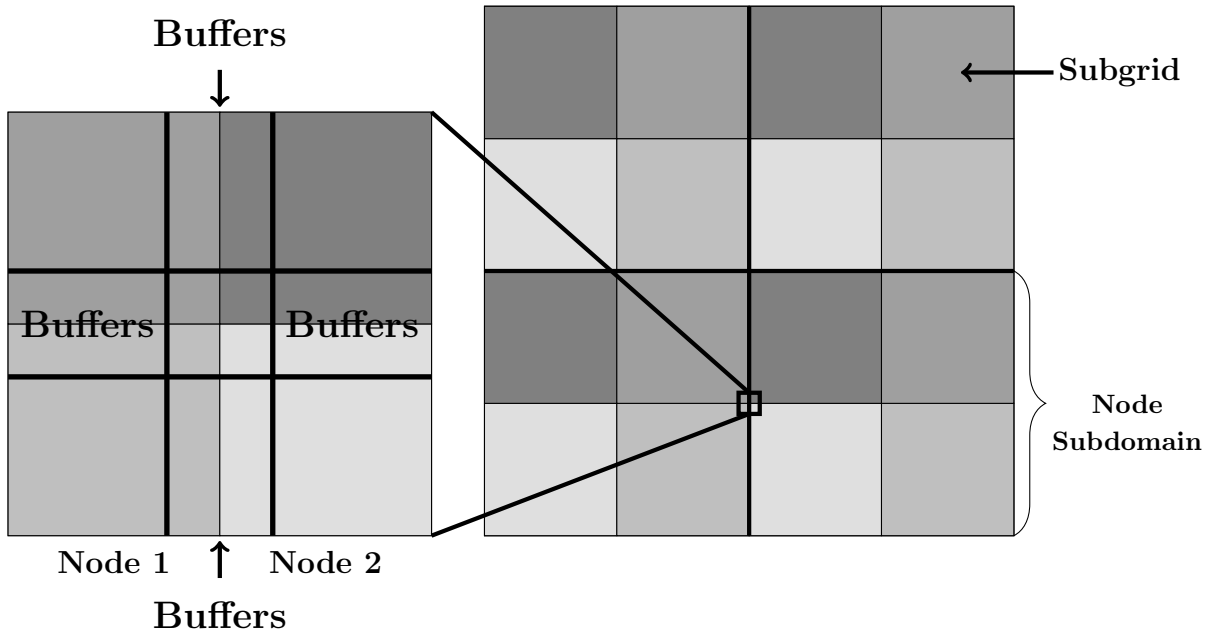


Figure 4.2: Domain decomposition in the parallel algorithm for the cellular Potts model. Recreated from Chen et al. [2007].

The FLAME software, reported in Chapter 2, provides functionality to simulate general agent-based models in parallel by creating parallel simulation code from the definition of agents and rules in an XML input file [Coakley et al., 2012]. The software can generate simulation code for the state machine model described in the context of epithelial cells Section 2.2.3, although these models may be applied to more general agent-based modelling problems beyond biology. Each agent has a number of state variables and rules

may be specified to define how an agent changes its state. In this paradigm agents only communicate through shared message boards which they may read from in order to obtain data for rule decisions, and write to based on its updated state. This structure provides an inherently ‘parallel’ approach, where agents can be updated asynchronously, provided the correct messages are available. By implementing the message boards in parallel using Message Passing Interface (MPI), Coakley et al. [2012] obtained 80% parallel efficiency on over 400 processors for a benchmark simulation of a simplified 2D molecular dynamics type system, using an AD approach to dividing agents equally between processors. Additionally, Holcombe et al. [2012] report a six times speed-up in the simulation of a model of foraging ants on 32 processors. FLAME also provides the possibility of decomposing agents based on their state variables (for example, a space decomposition) although the authors [Coakley et al., 2012] note that this approach gives rise to load-balancing problems, where different compute nodes may be responsible for a disproportionate amount of computation.

In tune with their increased use in MD simulations reported in the previous section, recent development in parallel cell population simulations has also focused on the use of GPGPUs. They are a popular choice, requiring modest capital outlay and achieving significant performance improvements over serial CPU algorithms. Although relatively limited by HPC standards they can make simulation for the individual scientist or group much more manageable, reducing simulations that took days to compute to an overnight task. Christley et al. [2010] have used the GPGPU approach to substantially improve the performance of the ScEM of Newman [2005]. By outsourcing computationally costly tasks, such as neighbour identification, to the GPGPU they were able to achieve an 18-fold increase in performance over the reported computation time by Newman. Nevertheless, they were only able to report on simulations where the total number of cells grew to approximately 5000, and such simulations took almost 10 hours to compute. The authors conclude that studies on the scale of 10^4 cells are not possible. Although simulating a single cell in the ScEM is more computationally intensive than in the model we initially

consider for parallelisation, the simulation of a single subcellular element is a similar task to simulating a single cell in the model from Chapter 3. A subcellular element is represented as a point in space which interacts with its near-neighbours, as is the case for an individual cell in our model. Christley et al. [2010] use approximately 20 subcellular elements per cell, and we may crudely extrapolate that a similar implementation of our centre-based model may achieve a limit of 2×10^4 cells, before memory limitations begin to become a factor in this approach. Richmond et al. [2010] implement an agent-based model of cell on a GPGPU in the FLAME software using a SD approach, and obtain large speed-up of 10,000 on an exemplar simulation. While 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.

In summary, success in using HPC technology for studying MD does not seem to have been replicated in the field of off-lattice computational modelling of populations of cells, in agreement with the concerns of Bader and Aluru [2008]. The most progress has come in the simpler lattice-based paradigm, where the greatest numbers of cells have been simulated. Some authors have 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 general purpose GPGPUs are of greater benefit [Alberts et al., 2011]. However, GPGPU technology, while providing valuable speed-up for relatively small simulations, lacks the memory and processing power to tackle new scales of problem. Chen et al. [2007] have shown that distributed HPC methods are a viable option for large-scale simulation of cell population models. In the next section we describe our approach to using this technology to enable larger and faster simulations of the off-lattice model.

4.3 Distributed algorithm

In the following sections we outline the steps taken to develop the distributed algorithm by first specifying how we choose to decompose the data and then reporting the necessary alterations to the serial implementation to correctly simulate an instance of the model.

4.3.1 Data decomposition

We choose to use the SD method for data decomposition, described in Section 4.2, because our principal goal is to enable the simulation of large problems. Decompositions using replicated data methods, where each processor stores data on every cell across the entire simulation, limit simulation size to the number of cells that can be stored in the memory of any single processor. In comparison, in the SD method each processor only stores those cells that lie in, or close to, its spatial domain. This should allow a greater number of cells to be simulated.

For generality we describe the decomposition of data assuming a 3D model; the reduction to lower dimensions is discussed briefly at the end of this section. The algorithm first calculates an axis-aligned bounding box for the initial locations of each cell centre, which we denote by $\Omega = [x_{min}, x_{max}] \times [y_{min}, y_{max}] \times [z_{min}, z_{max}]$. The initial state of each cell is stored on all processors, and only data on those cells that will belong to the local processor are added to the simulation data structures. The fact that the entire initial condition is stored on all processors restricts the size of the initial condition to the local memory size, a problem we discuss and overcome in Chapter 5.

The cell-list method, used in the serial algorithm, provides a natural decomposition of the spatial domain. We briefly recap this approach for clarity. Given a maximum cell-cell interaction distance r_c , the domain Ω is divided into cubes $C_{i,j,k}$ of the form

$$C_{i,j,k} = [x_{min} + ir_c, x_{min} + (i+1)r_c] \times [y_{min} + jr_c, y_{min} + (j+1)r_c] \times [z_{min} + kr_c, z_{min} + (k+1)r_c],$$

where $i = 0, \dots, \lfloor (x_{max} - x_{min})/r_c \rfloor$, and equivalently for j and k . Figure 4.3 shows such

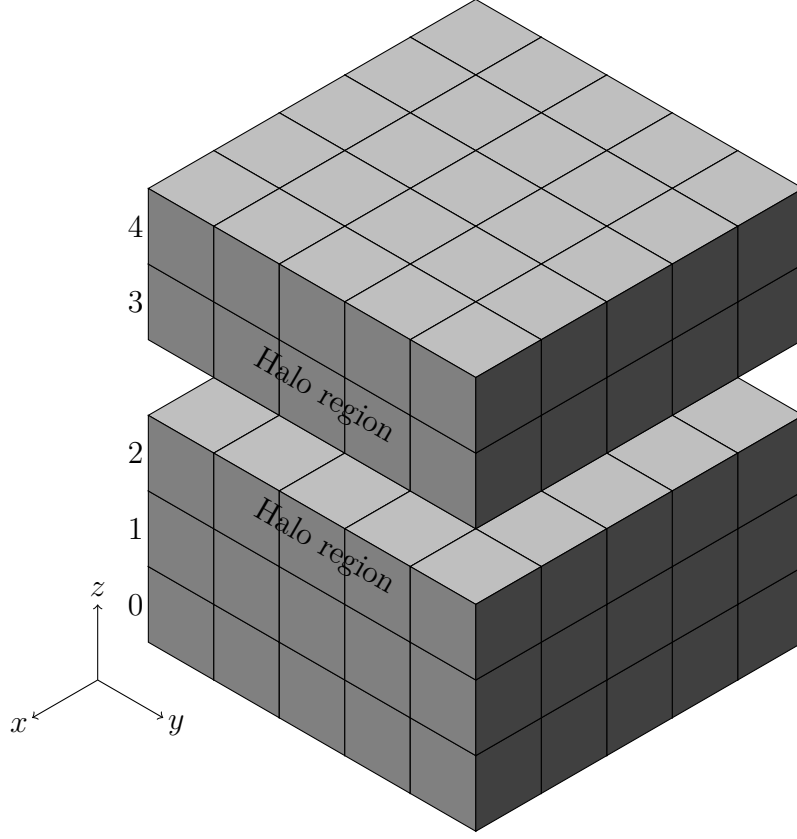


Figure 4.3: Domain decomposition approach in 3D. The domain is divided into cubes of linear dimension equal to the interaction cut-off length. The layers are distributed between the processors using a distributed vector, which is shown represented by the integer values 0 – 4 adjacent to the cubes.

a splitting of the spatial domain. This approach will actually overshoot the domain by a small amount in each co-ordinate direction, and consequently we swell the initial domain size such that it is divisible by the cut-off distance, r_c in each direction.

To distribute the cubes, and hence the spatial domain, between processors we use a distributed vector class from the PETSc library [Balay et al., 2011]. A distributed vector assigns an equal (or as close to equal as possible) and contiguous subset of the indices of the vector to each processor in a distributed environment. We create a distributed vector containing the entries $(0, 1, \dots, (z_{max} - z_{min})/r_c)$, and assign each processor the space in all those $C_{i,j,k}$ such that k lies in the local portion of the distributed vector, i.e. the domain is split by planes orthogonal to the z -axis. Figure 4.3 shows how this works for a small example. Each processor then stores those initial cells that lie in its region of

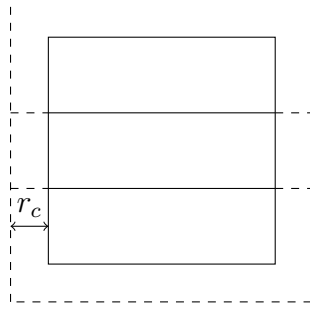


Figure 4.4: Approach to enlarging the spatial domain for a growing simulation that ensures the space decomposition is constant. Old domain shown by the solid line, new domain shown by the dashed line.

the domain, and discards all others.

In contrast to the SD approach described in Plimpton [1995] we divide the domain by planes rather than in to individual cuboids. We choose this approach to limit the number of different neighbouring processors that each processor will share a boundary with, so that the impact of requiring synchronisation between processors during communication may be limited. In order to minimise the total volume of communication between processors, a cuboidal decomposition may provide a better solution, at the cost of increasing the number of communication messages each processor must create and decode.

We allow the domain to dynamically grow with the size of the population, so that it is not necessary to specify a domain size when setting up a model with free boundaries. At each time iteration each processor checks that none of its cells are closer than the interaction cut-off, r_c , from the boundary of the domain. All processors then exchange a boolean value with all other processors to indicate if they own a cell close to the boundary. If any processor has a cell close to the boundary the domain size is increased by r_c in each co-ordinate direction. The same distribution of space is used, with the top and bottom processors gaining an extra plane of cuboids, as shown in Figure 4.4 for the 2D case. This means that no cells will be reallocated to different processors when the domain is enlarged.

4.3.2 Halo data and cell migration

If each processor were to execute the serial algorithm on the local cells stored in its memory, the simulation would not be a faithful implementation of the model: cells that satisfy the neighbourhood criterion but that lie on different processors would not exert a force on each other. In order to overcome this, each processor must communicate cell location data with the processors owning space adjacent to its own domain (neighbouring processors). To minimise the amount of communication, we only communicate cells that lie within a distance r_c of the boundary with another processor (and hence may interact with cells on neighbour processors). This equates to communicating cells that lie in the cubes adjacent to the processor domain boundary. We use the term *halo cubes* to describe those cubes that lie on neighbouring processors, and that are adjacent to the boundary. The cells that lie in *halo cubes* on an adjacent processor are termed *halo cells*.

When cell neighbours are calculated in the distributed algorithm, halo cells are used as well as local cells, so that the force exerted on a local cell by a cell stored on another processor can be calculated. The corresponding force on the halo cell is not considered and the halo cell is not moved, since its motion will be calculated by its owning processor. The halo cells are only stored for one time step, after which they are completely refreshed.

This modification of the serial algorithm ensures that the simulation is correct for the first time step. However, the method of collecting halo information only from neighbouring processors will lead to errors if a cell moves out of the domain of the processor to which it was initially assigned. Therefore we must ensure that a cell whose location moves over a processor boundary is transferred to the processor owning its new location. We call this operation cell migration. Since, at each time step, the forces on every cell are calculated before any of the cells are moved, it is possible to carry out cell migration after the location of all cells have been updated, in one synchronous step, without introducing errors into the simulation. The new processor stores re-allocated cells alongside local cells and the cell is deleted from the memory of the processor from which it is migrated.

The modifications described above constitute the principal changes to the serial algo-

rithm. However, the implementation of these changes into Chaste gives rise to a number of issues, which we describe in the following sections.

4.3.3 Data communication

To communicate cells between processors, we use an implementation of the MPI for C++ [Gropp et al., 1999]. MPI is a model for communication in a distributed-memory computer. It is implemented as a library for C++, providing functions to copy the contents of sections of memory between processors. A primary advantage of using MPI is that its implementation can be used on shared-, as well as distributed-memory multiprocessor architectures. However, standard MPI implementations only provide methods for communicating contiguous portions of memory, containing a single data type such as a `char` or a `double` and simple extensions of these types. In MD simulations these methods are enough to communicate locations of particles, which are vectors of floating-point data. In contrast, cell-cell interactions can depend on many of the state variables of the cells and we therefore need to communicate much more data than just a cell's location. Furthermore, the extensible nature of the implementation of the model in Chaste means that cell-cell interactions may depend on any combination of cell state variables. This means that basic MPI methods are insufficient on their own to communicate cells between processors.

A number of options can be considered to solve this communication problem. One, perhaps initially appealing option is to send the least amount of data necessary to ensure that the model is simulated accurately. For halo cells this constitutes the location of the cell and any data associated with it that affects its interaction with its neighbours, such as its radius. In the case of migrated cells we would have to send all the data associated with the cell to reconstruct it accurately on the neighbour processor. In practice this is difficult to do, since a cell consists of many different data structures. Moreover, the hierarchical structure of classes in Chaste means that although every cell contains data of the same base types, each piece of data can be of a wide variety of child types, each with

their own varying data structures. Therefore, a new communication method would have to be written for every permutation of possible types of cells in Chaste, and a further method written for every different type of cell neighbour interaction. This approach does not conform to the Chaste theme of developing extensible, modular code and would mean new users would need to create new communication methods. A neater solution is to create a helper class containing methods that can send or receive any type of cell object.

To implement a class that can send or receive any type of cell object, we take advantage of the fact that all of the classes used as data types in a Chaste cell contain methods to write their data to a file in a structured way, so that a particular cell object may be re-created by the cell class at a later time (often called ‘archiving’, ‘serialisation’, ‘checkpointing’ or ‘pickling’)². Chaste uses the Boost Serialization library to archive its classes [Karlsson, 2005].

While this pre-existing functionality provides the basis for communicating instances of the cell object, it is necessary to make a number of extensions to enable its use in practice. Firstly, instead of writing to file, we write the archive to a memory buffer as a string of characters. This string of characters is stored in a contiguous piece of memory and may therefore be communicated using basic MPI methods. Once received, the string of characters can be re-formed into the archive and restored using an instance of the class on the receiving processor. Libraries with similar functionality, that use serialisation techniques to make communication of complex data structures through MPI easier, are available in the form of Boost MPI, and also the ClassdescMP library [Standish and Madina, 2003]. However, we chose to write our own class to handle this functionality so that we did not introduce a further dependency into the Chaste library. Our helper class, called `ObjectCommunicator`, encapsulates the functionality described here, and is available as part of the Chaste source code.

²Most of the classes in Chaste can be archived using this method.

4.3.4 Cell identifiers

In simulating an instance of the cell population model it is necessary to be able to uniquely identify each cell over the course of a simulation. We may wish to track its motion, or to select the cell to die or divide. This necessitates having a unique identifier for each cell, which we achieve by having each cell store a unique positive integer. In a serial algorithm this is simply a case of: assigning the initial n cells the identifiers $0, 1, 2, \dots, n - 1$ in an arbitrary order; keeping track of identifiers that are freed by cell death; and assigning a free identifier (either from a dead cell, or choosing the smallest unused integer) to one of the daughter cells after cell division, with the other daughter keeping the identifier of its parent. For simplicity, in serial the identifier is chosen to be the index of the corresponding **Node** in the vector **mNodes**, which stores location data for each cell, as described in Chapter 3.

This approach is a poor choice for the distributed algorithm. Its use would require additional communication to ensure unique identifiers are used on all processors. On each processor it would be necessary to track all the used identifiers across the entire simulation, and this list would need to be updated at every time step. Furthermore, the algorithm would need to ensure that two cells born simultaneously on different processors were not assigned the same identifier. Since the actual value of the identifier is arbitrary, we choose to alter the serial algorithm, and have each processor generate identifiers by striding over the integers with a step of size equal to the total number of processors. The identifiers for each initial cell on processor with rank³ r are chosen from the set containing positive integers that are equal to r , modulo the total number of processors. A cell that is migrated from one processor to another keeps its identifier, and its original parent processor stores a list of identifiers of cells that have migrated to other processors, so as not to re-assign them to new cells. When a cell is born it is assigned the smallest unused integer that is equal to the processor rank, modulo the total number of processors. One feature which is

³A parallel environment with k processors assigns to each processor a unique integer in $0, 1, 2, \dots, k - 1$, called the processor rank.

lost from the serial algorithm is that the cell identifier no longer corresponds to the index in `mNodes` of the node associated with each cell. To account for this, for each processor, we create a map from the cell identifier to its index in the local `mNodes` vector and *vice versa*. This enables identification of the location data for each cell, and for a cell to be obtained using its location data. When cells are migrated, or birth or death occur, the map is updated. The memory cost of this is small and scales with the number of local cells. Although this method introduces additional computation, we assume this to be insignificant compared to the complication and communication cost of maintaining the serial approach for cell identifiers.

4.3.5 Output

Visualisation and post-processing of the results for analysis of a simulation requires the data for each cell to be written to file. The serial approach is to open a file and write the data for each cell in turn at each time step before closing the file at the end of the simulation. However, in parallel it is not possible for each processor to write to the same file synchronously since one processor may overwrite another processor's output. Reducing all data onto one master processor that writes results, or passing a results file from processor to processor, are both options that scale poorly. The former suffers from extremely high communication latency for large populations of cells, as a vast amount of data must be communicated, while the latter has a lower communication cost but does not allow synchronous writing of results, so while one processor is writing, every other processor must wait.

Our solution is to use methods for writing Visual Tool Kit (VTK) files in parallel [Schroeder et al., 2006]. This approach allows each processor to write results to its own output file, at each time step. A short master file is then written, which contains information on how these files should be combined for visualisation in tools such as ParaView [ParaView, 2012]. We find that this is a simpler and more scalable approach, since most of the functionality for writing these files was already implemented in Chaste before

the commencement of this project.

In order to maintain the functionality in Chaste to write raw data to text-based files, we also implement a round-robin system where the processors write their data to a single file one after the other. While this is less efficient, as each processor has to wait while all others write data to file, it is useful for data analysis and important to maintain legacy functionality. To maintain efficiency of our algorithm we provide the option to switch this output off and use VTK exclusively.

4.3.6 Simulation in fewer spatial dimensions

We have described the algorithm for simulating our model in 3D. To simulate reduced models in 2D, for example to study the growth of a monolayer of cells grown on a Petri dish or other flat substrate, the only modification is dividing the domain into squares, rather than cubes, and splitting the domain perpendicular to the y axis, rather than the z axis. All of the other features of the algorithm transfer directly in to lower dimensions.

4.4 Results

4.4.1 Validation

We validate the distributed algorithm and its implementation by comparing results of simulations run in parallel with those obtained using the serial algorithm. Ideally we would compare our results to an analytic solution for the evolution of the model, but the complexity of the model renders calculating such a solution an impossibility for any non-trivial simulation.

We consider two types of validation simulations: unit tests (unit tests are described in Section 3.3.1) that simulate the evolution of a small number of cells under different conditions, and a deterministic simulation of a larger population of cells. The former allows us to validate the algorithm for the calculation of forces between cells that are located on different processors, and the case where a cell is pushed over a processor

boundary. The latter test provides the closest approximation to a full-scale cell population simulation that can be practically validated using this method. A model that is not deterministic, i.e. has stochastic elements such as the division axis of a cell, is difficult to validate, since its results will depend on how random numbers are generated on each processor.

We validate the numerical output of the cell-level model, that is, the location of individual cells at each time step. These data are written to file in a round-robin fashion as described in the previous section which means that the order of the written data, with respect to the cells, can vary depending on how many processors are used. To overcome this, we compare the data for each time step from the serial and parallel results files once they have been (lexicographically) sorted. Our metric for comparing the sorted data files is

$$\sigma_n = \max_i \|\mathbf{x}_i^s(t_n) - \mathbf{x}_i^p(t_n)\|_2,$$

where $\mathbf{x}_i^s(t_n)$ is the recorded location of the i^{th} cell at time step t_n , from the serial algorithm and $\mathbf{x}_i^p(t_n)$ is the recorded location of the i^{th} cell at time step t_n from the parallel algorithm. For the unit tests carried out, it was confirmed that $\sigma_n < \epsilon_{\text{machine}}$ for all n , where $\epsilon_{\text{machine}}$ is the difference between 1.0 and the next floating-point number. The value $\epsilon_{\text{machine}}$ is an appropriate threshold since all tests are of approximately unit size. On the test machine, a 64-bit Linux desktop, $\epsilon_{\text{machine}} = 2^{-52} \approx 2.22 \times 10^{-16}$. For the deterministic simulation we plot σ_n as a function of t_n .

4.4.1.1 Unit tests

We perform two unit tests, which simulate a model of two cells to verify that the equations of motion have been implemented correctly. In the first test the force between two cells whose centres are physically separated by less than the cut-off distance, but are stored on different processors, is calculated using halo cells. Cells are initially placed at (0.0, 0.0, 0.0) and at (0.0, 0.0, 0.4) on processor P_0 , and another is initially placed at (0.0, 0.0, 0.6) on



Figure 4.5: Three cells (owned by separate processors) push apart as a result of a linear spring force interaction. The initial conditions on (a) one and (c) two processors and the final state on (b) one and (d) two processors. Cells are coloured according to which processor owns them.



Figure 4.6: Three cells push apart as a result of a linear spring force interaction. The initial conditions on (a) one and (c) two processors and the final state on (b) one and (d) two processors. Cells are coloured according to which processor owns them.

processor P_1 . The additional cell at $(0.0, 0.0, 0.0)$ ensures the domain is divided between processes with each of the other cells lying on different processes. The force between the cells is calculated using the quasi-linear spring force, as described in Chapter 3. Figure 4.5 shows a visualisation of the initial and final state of the cells for the one- and two- processor tests and we see that the solutions are identical.

The second unit test has a similar setup to the first, but all cells initially lie on P_0 . As the simulation proceeds, the spring between the cells relaxes, which pushes the cell at location $\mathbf{x}_1(t_n)$ over the boundary between P_0 and P_1 . Figure 4.6 shows a visualisation of the initial and final state of the cells for the one- and two- processor tests.

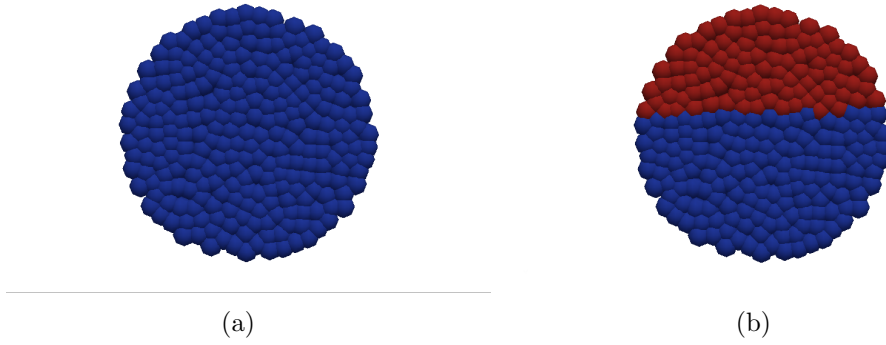


Figure 4.7: A population of cells interact through the linear spring force law on (a) one processor and (b) two processors.

4.4.1.2 Deterministic simulations

While unit tests allow us to validate components of the implementation of the model, they cannot represent a full-scale multi-cell simulation. Problems could arise, for example, if small numerical discrepancies between the serial and distributed implementation occur. In a large-scale, long-time simulation these errors could propagate to significantly alter the results of the simulation. While the scale of the problem dictates that we cannot compare our implementation with the serial version for the size of model we wish to simulate, we can validate moderately large populations of hundreds to thousands of cells.

It is necessary to remove stochasticity from the simulation for the results of the serial and distributed implementations to be comparable. Without a burdensome level of communication between processors it is not possible to ensure that (pseudo-) random numbers are generated in exactly the same sequence in a distributed environment as they were in a serial environment. We instead choose to fix, for the purposes of these validation tests, the response to all requests from a simulation for a number drawn from $\text{Uniform}(0, 1)$ to return 0.5. For this test an initial condition of 256 cells in 2D are arranged within a circle and simulated in accordance with our model. The population is allowed to evolve and the forces relax. In Figure 4.8 we plot $\sigma_n/\epsilon_{\text{machine}}$, and observe that the difference between the numerical output between the two simulations is on the same order of magnitude as the machine precision.

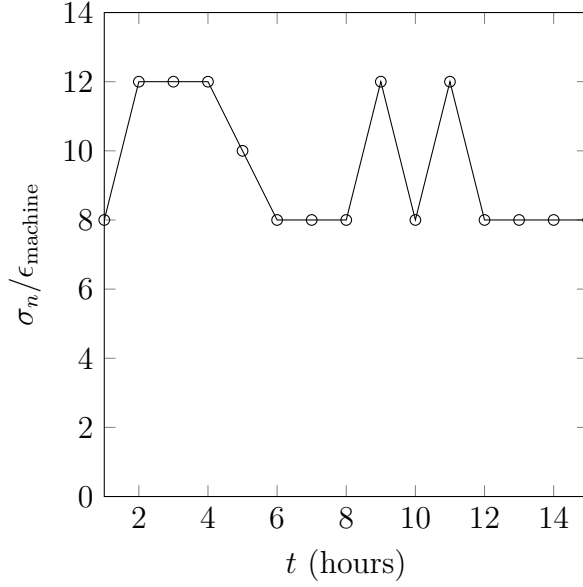


Figure 4.8: Maximum difference between cell locations from the serial and parallel simulations relative to machine precision for a deterministic cell population simulation.

4.4.2 Simulation of a simplified cell population model

To obtain a benchmark for the performance of our distributed algorithm, we first test it on a simplified model. We consider an abstract model of mechanical cell-cell interactions and random motion. However, we simplify the model by removing all sub-cellular processes such as the cell cycle model and mitosis. This greatly simplifies the necessary communication between processors; moreover, implementation in a stand-alone C++ program removes the complex data structures of Chaste, making the simulation more akin to a simple MD simulation. The details of the force laws governing cell-cell interactions are identical to the full model. Random motion is modelled by each cell taking a random jump of length $\sqrt{2D\Delta t}$ (where D is a tuneable parameter and Δt is the integration time step) in a random direction at each time step. Here we present results showing approximate linear scaling of the computational cost of the simulation with the number of processors used, and how the scaling varies with the speed of the random motion.

We performed the simulations on between one and eight processors, situated on the same node of a cluster. Each processor is an Intel Xeon CPU with 2.80Ghz clock-speed.

MPI communication is carried out using shared-memory with the MPT implementation of MPI. We chose to restrict to those processors available on a single node so that communication latency would be approximately symmetric between all processors. Communication between processors on different compute nodes suffers from higher latency, which would create incomparable scaling results. Expansion to larger numbers of processors is discussed in Section 4.5.

Throughout this section we will use *speed-up* as the metric to describe the increase (or decrease) in performance of a distributed algorithm as it is executed on an increasing number of processors. For a simulation which consumes T_p s of wall time on p processors, speed-up on p processors is defined as T_1/T_p .

Figure 4.9(a) shows the speed-up for the abstract cell population simulation. The data closely match ‘linear speed-up’, which is the theoretical limit for speed-up of a parallel algorithm. It is possible, in some cases, to exceed this limit due to the more efficient use of the computer architecture in parallel; for example when data is distributed between processors a greater proportion of the data may be stored in fast-access cache memory. Such a high degree of speed-up is only possible if the increase in the level of communication between successively greater numbers of processors is outstripped by the benefits of distributing the computation between them. In Figure 4.9(b) we can see that varying parameters of the model can affect this parallel efficiency. By increasing the magnitude of the random force applied to each cell at each time step, the cells have a much greater average speed and consequently a greater number cross between processor boundaries every time step. This leads to more communication between processors, which can be seen to decrease the speed-up.

We expect these results to be translated into a parallel algorithm for the full model. That is, we can expect good scaling for simulations of populations of cells that move a small distance (compared to their cell radius) over the course of a simulation, such as cells in compact tumours with minimal cell division. We would expect to see substantially poorer speed-up results for simulations where cells undergo large displacements, such as

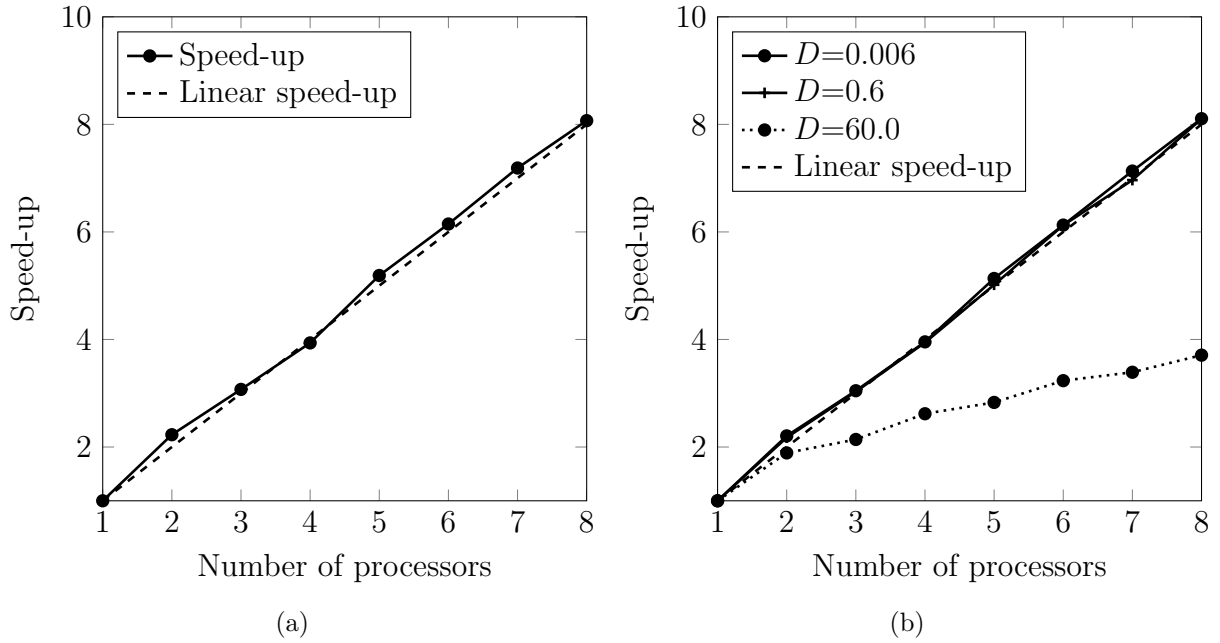


Figure 4.9: (a) Approximate linear scaling is achieved for a simulation of the 2-D cell model using the distributed algorithm with $D = 0.006$. (b) Variation in the magnitude of the diffusion force, D , can affect scalability by increasing the volume of cells being migrated between processors. The lines for $D = 0.006$ and $D = 0.6$ are almost coincident.

those used for free-swimming bacteria.

4.4.3 Implementation of the full model

To test how well our implementation performs when applied to the full model in Chaste, we carry out four sample simulations that either model relevant biology or demonstrate the implementation's strengths and weaknesses. We first simulate a static cube of 125,000 cells, which do not move in response to their neighbours, but still calculate the neighbourhood topology. This problem is not intended to approximate a concrete biological system, but instead it serves to demonstrate the optimal performance of our implementation since the cells fill the domain, and no cells are migrated between processors. We shall later see that when these two conditions are not met, the algorithm performance decreases. We also consider two typical applications of our cell population model: a monolayer of cells growing on a flat substrate, and a 3-D aggregate of cells growing in a

nutrient rich environment. Finally we carry out a test analogous to that shown in Figure 4.9(b), where each cell performs a random walk of constant step-size through a 2D domain $\Omega = [0, 300] \times [0, 300]$ imposed using hard-wall boundary conditions, experiencing adhesive-repulsive forces with neighbouring cells in accordance with the model described in Chapter 3. To implement the random walk, we impose a force $\eta_i \sqrt{2D/\Delta t} \boldsymbol{\xi}_i$ to each cell i at each time step Δt , where η_i is the cell damping constant, D is a tuneable parameter and $\boldsymbol{\xi}_i$ is a vector such that $(\xi_j)_i \sim N(0, 1)$. The cells thus move, on average, a distance $\sqrt{2D\Delta t}$ at each time step. To ensure that motile cells cannot move outside of the domain, we impose hard-wall boundary conditions: if a cell crosses the boundary, it is immediately moved to the boundary line. For the performance tests, we vary the value of D to see how increased migration of cells between processors will affect a simulation of a highly motile or diffusive versus a relatively static cell line. In Figure 4.10 we visualise each of these five simulations using ParaView [Henderson, 2007], showing the spatial configuration of the cells, and where appropriate a super-imposed time course to show the growth of the population.

Figure 4.11 shows the performance results for each of these simulations. The optimal simulation of a static cube of cells (Figure 4.11(a)) initially performs well for a few processors, but the perfect linear speed-up achieved by the abstract cell population simulation is not reproduced. The speed-up is further limited when we consider a growing monolayer of cells (Figure 4.11(b)). Performance below that of the static cuboid is mostly due to the simulation beginning from a small seed of cells: the early stages of the simulation will only have cells stored in a small sub-set of the total processors. Therefore potential speed-up is lost as processors are idle for a large proportion of the simulation. This inefficiency is amplified in the 3D population (Figure 4.11(c)). The simulation is in fact slower in parallel for a growing 3D population, than it is in serial.

Figure 4.11(d) presents scaling for the simulation of cells performing a random walk. The parameter D is set to 60.0 and 0.006 for the motile and non-motile cells respectively. We see scaling similar to that for the abstract model: that is, the simulation of non-motile

cells has better scaling than the simulation of motile cells. However, the performance difference is significantly less pronounced.

To understand the cause of the limited speed-up obtained when our algorithm is applied to the full model, we plot data on the proportion of computation time taken by each algorithm task in each of the exemplar simulations. Figure 4.12 presents these data where each line is wall time spent in:

- ‘Cell birth’ – dividing cells into two daughters;
- ‘Cell migration’ – resizing domain and communicating cells that have changed processor;
- ‘Halo cells’ – updating the halo cell data;
- ‘Remesh’ – reorganising local data structures to minimise memory use;
- ‘Cell neighbours’ – calculating neighbouring cells;
- ‘Cell force’ – calculating the forces on each cell;
- ‘Cell position’ – updating the location of cells.

Wall time for a task is defined as the elapsed time from the start until the end of a computational task. In Figure 4.13 we also show the balance of the computational cost of cell neighbour calculation between each processor. This provides an indication of how evenly the local computational tasks are distributed between the processors, since cell neighbour calculation is the most expensive part of the code.

Figure 4.12(a) suggests that the cuboid simulation also scales poorly due to increased communication. The proportion of time communicating halo cells grows as the number of processors is increased. The speed-up achieved is a result of decreasing the cost of the, initially dominating, cell neighbour calculation. Figure 4.13(a) shows that the local computational tasks are, however, well distributed between the processors since the cells exactly fill the domain.

Growing populations in the monolayer (Figure 4.12(b)) and spheroid (Figure 4.12(c)) simulations are dominated to an even greater extent by the communication operations. Similarly to the cuboid example in Figure 4.12(a), we also see growing costs of communicating halo cell data. Furthermore, we note a sharp increase in the cost of ‘Remesh’ using 3 or more processors. This is as a result of load-imbalance between processors where all processors are forced to catch up at this point. This can be more easily seen in Figures 4.13(b) and 4.13(c). The computational task of simulating a growing population is poorly load-balanced using our algorithm, with much of the computation occurring on the processors owning the extremal regions of the domain which have extra space allocated to them as the population grows.

Figures 4.12(d) and 4.12(e) show the division of work for the simulation of motile cells ($D = 60.0$) and non-motile cells ($D = 0.006$). There is little difference in the division of work, and this suggests the cause of the limited speed-up. The overhead associated with communicating data is the dominant factor in each simulation, and potential gains from small reductions in migration are dominated by the cost of the communication that does occur. We see in Figures 4.13(d) and 4.13(e) that these simulations are well load-balanced, because the cells are evenly distributed through the domain for the entire simulation.

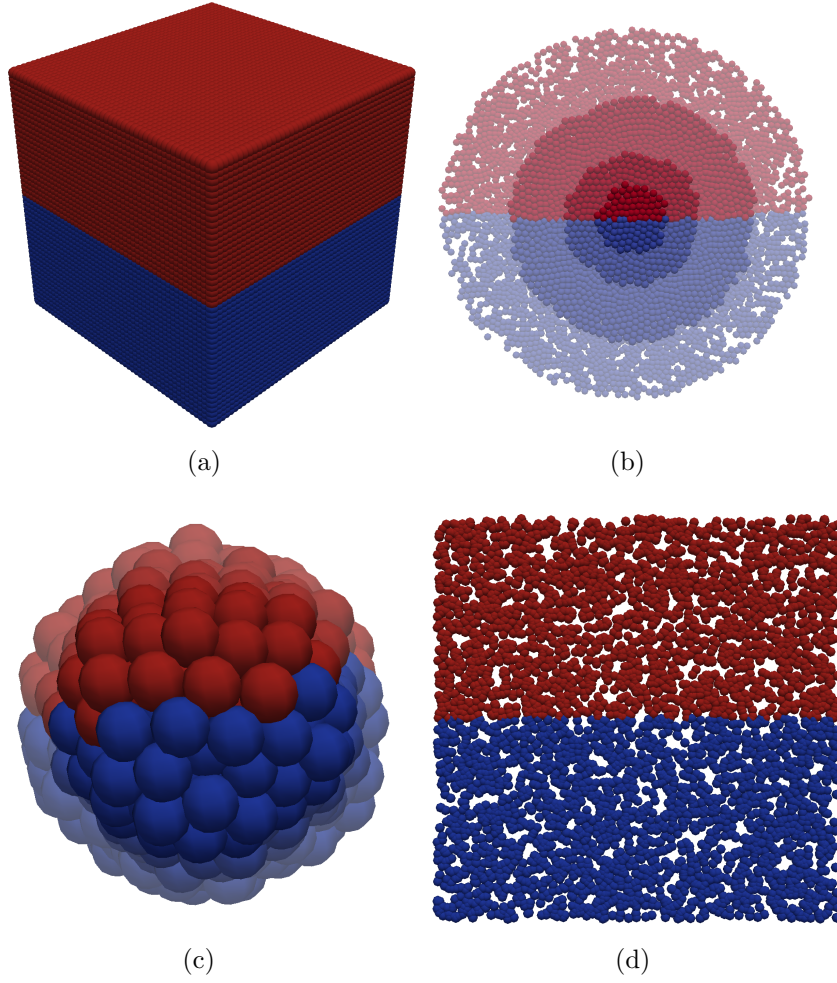


Figure 4.10: Visualisation of the five different simulations used to benchmark the parallel algorithm. (a) Static cuboid of 125,000 cells; (b) growing 2D monolayer (shows $t = 5, 25, 50, 100h$ super-imposed); (c) growing 3D spheroid (shows $t = 1, 5, 10, 15h$ super-imposed); (d) motile cell population and im-motile cell population.

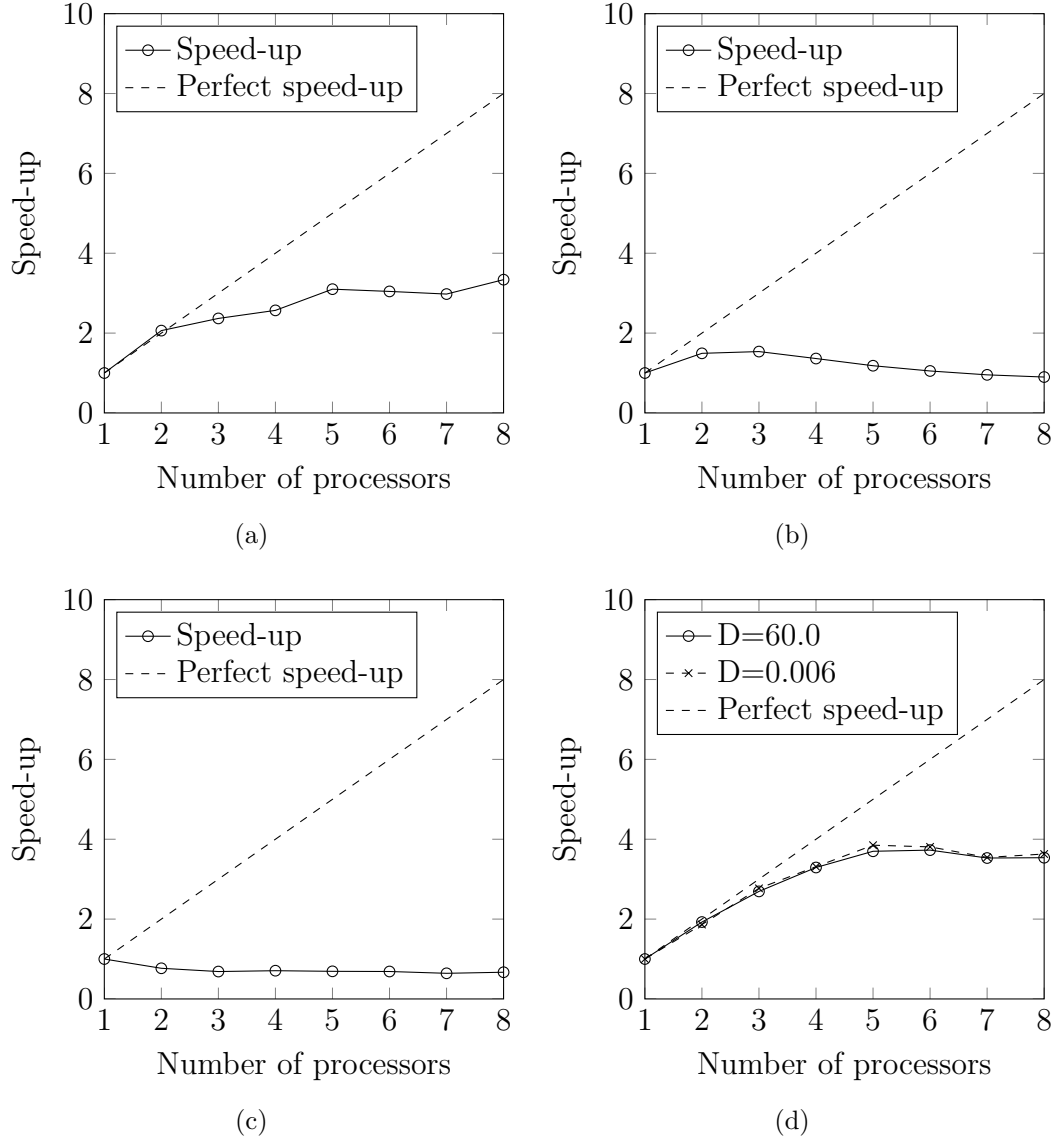


Figure 4.11: Speed-up results for simulations of the full cell population model implemented in Chaste. $D = 0$ unless otherwise stated in figure legend. (a) Static cuboid of 125,000 cells. (b) growing 2D monolayer. (c) growing 3D cuboid. (d) motile cell population and immotile cell population, characterised by the parameter D .

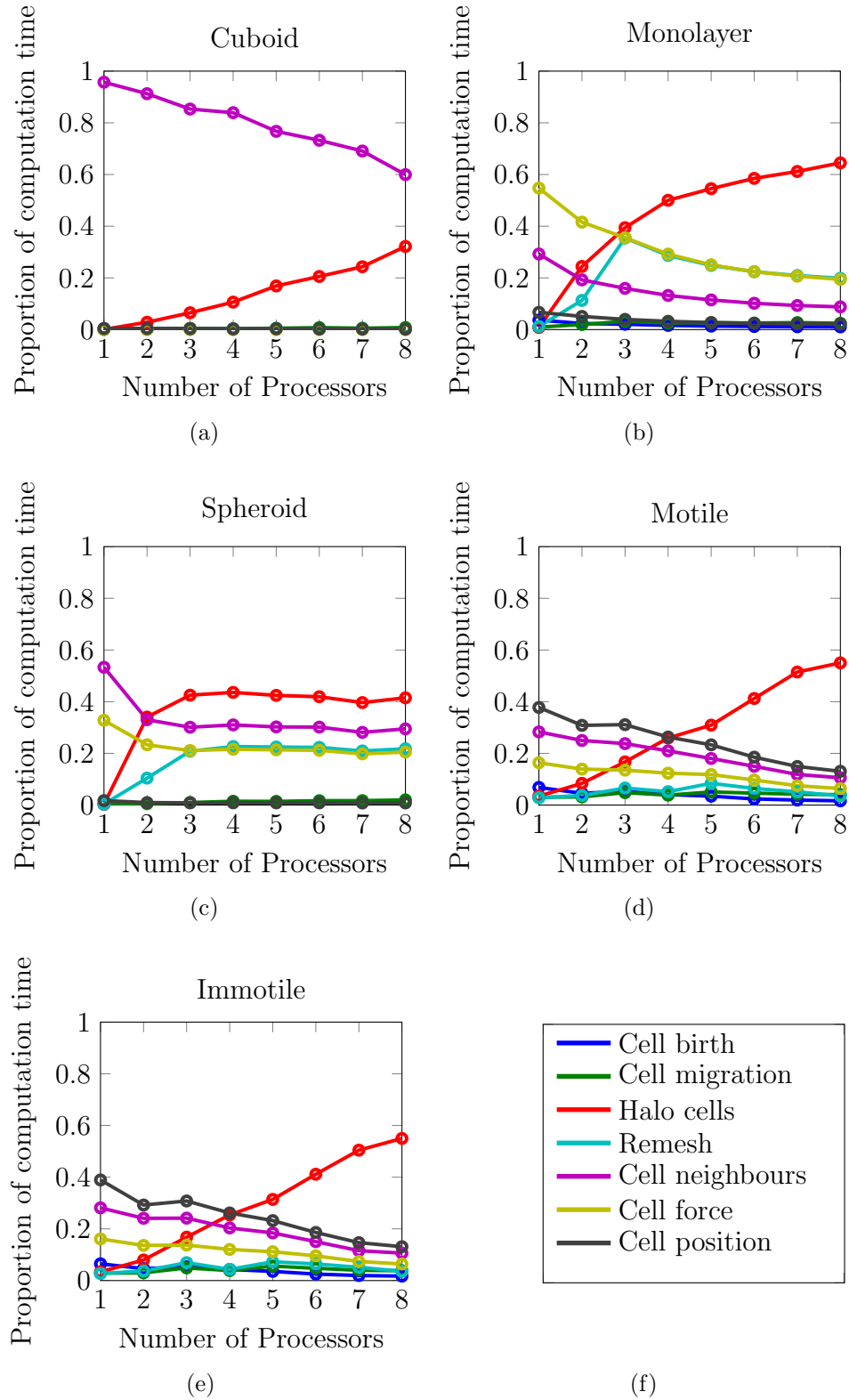


Figure 4.12: The proportion of computation time spent on each algorithm task for simulations run on 1 – 8 processors. (a) Static cuboid of 125,000 cells; (b) a growing 2D monolayer; (c) a growing 3D spheroid; (d) motile cell population; (e) immotile cell population.

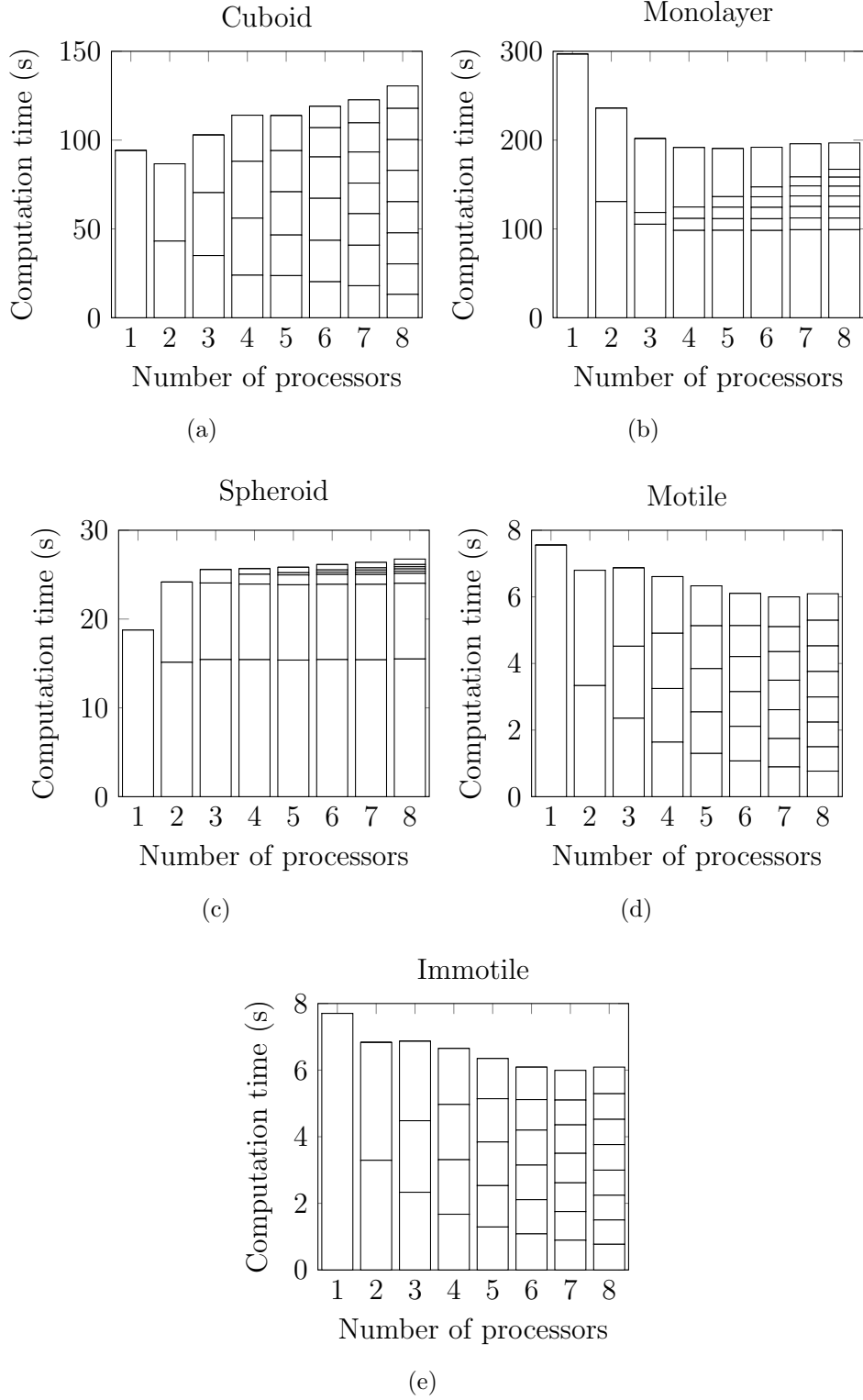


Figure 4.13: The distribution of computation time between each processor for the calculation of cell neighbours. (a) Static cuboid of 125,000 cells; (b) growing 2D monolayer; (c) growing 3D spheroid; (d) motile cell population; (e) immotile cell population.

4.5 Discussion

In this chapter we have described and tested a novel distributed algorithm that uses space decomposition to simulate the model described in Chapter 3 with the aim of enabling a broad range of larger-scale simulations to be performed. By choosing an SD approach we have minimised the total volume of data that must be communicated. Consequently, with the exception of large initial conditions, we have solved the memory bottleneck imposed by storing all cell data on a single processor. Our implementation of inter-processor communication of individual cells is extensible so that many model variants can be easily set up and simulated in parallel without the need for further development of the algorithm. We have validated the results of the algorithm by showing that they do not differ from those produced by the serial algorithm.

The use of serialisation techniques to communicate cell data between processors means that our parallel implementation is extensible. This is a valuable property since future development of the model can be handled simply by our implementation. As we do not explicitly specify the data to be sent for each cell, relying rather on the serialisation methods built into classes in Chaste, future models may have interactions that depend on any of the state variable of a cell. The cost of this approach is that, irrespective of the data needed to accurately simulate the model, all cell data is transmitted. It may be the case that such an overhead may be avoided for specific instances of the model for which cell-cell interactions only depend on a subset of the cell data. In such cases it may be possible to send a reduced subset of the cell data between processors and reduce the communication overhead of the algorithm.

By performing unit tests and deterministic simulations we are confident that the implementation of our algorithm in Chaste accurately simulates the cell population model. Since the serial algorithm is extensively tested, comparison with its results provides substantial evidence for the validity of our simulations. It is not possible, however, to test large-scale simulations, or simulations with stochastic elements such as cell division direction, or cell cycle time, using this validation method. Large simulations are intractable

in serial and, as discussed in the main body of the chapter, replicating the generation of sequences of pseudo-random numbers in parallel is beyond the scope of this work. A further extension of the validation process would be to set the viscous drag on each cell, given by the parameter η_i , to be zero, obtaining a closed system

$$m_i \frac{d^2 \mathbf{x}_i}{dt^2} = \mathbf{F}_i,$$

where the total kinetic energy in the system would be conserved. It would be possible to test that total energy is conserved in the parallel algorithm to further verify our approach. However, the assumption that m_i/η_i is small, and hence the dynamics of the system are dominated by the viscous term, is central to the integration scheme in Chaste, and the implementation of new integration schemes is beyond the scope of this thesis.

The results of scaling experiments for a selection of representative simulations reported in Section 4.4.3 lead us to believe that our choice of the SD method is likely to offer better scaling for our problem than the other data decomposition methods used in MD simulations which we described in Section 4.2.1.3. Our simulations are dominated by the communication costs as the number of processors is increased, and, since the other methods rely further still on communication, we believe that these approaches would suffer from even greater parallel inefficiency.

By implementing our algorithm for a simplified cell population, with only location data stored for each cell, we are able to achieve optimal parallel efficiency, but applied to the full model our approach does not scale so well. The efficiency of our algorithm for the full cell population model is reduced by the high cost of communicating cell data and poor load-balance for simulations of a growing population. Although our method for communication is extensible, this is at the cost of sending all cell data irrespective of whether it is needed to accurately simulate the model. Halo cells, for example, require only the data on which cell-cell interactions depend. Furthermore, the division of the spatial domain between processors takes no account of the spatial distribution of cells (which is not known *a priori*) and, consequently, the distribution of computational work between

processors. This approach provides good load-balance when cells are evenly distributed for the duration of the simulation, such as the models of motile cells, but applications where a population grows from a small initial seed, such as a study of a growing solid tumour, are not well load-balanced.

A further limitation of the current algorithm is that it is necessary for each processor to store the entire initial condition of the simulation and consequently the memory footprint grows with the number of processors used. This restricts the size of the problem that may be solved to those for which the size of the initial data are less than the capacity of the smallest memory unit in the distributed computer.

The future development of this algorithm revolves around these problems: high communications costs, poor load-balance for non-regular cell distributions and non-scalable memory use. In order to maintain the extensibility of our implementation we investigate a number of approaches to reducing the cost of communicating the halo cell data between processors. In Chapter 5 we first make the assumption that all processors in the distributed system have the same underlying architecture enabling faster serialisation of data. Further, we investigate the use of non-blocking communication, which enables other computational work to be carried out while data is transferred between processors.

The introduction of a dynamic load-balance approach will also improve simulations of growing populations. The most simple approach is to enable the decomposition of the domain to change in order to minimise the variation in the number of cells owned by each processor. This approach is also outlined in the next chapter.

To address the limits on the size of initial condition that may be considered, in the next chapter we develop a novel approach for constructing simulations from data files stored in large hard-disk memory, to ensure that the use of relatively limited random access memory scales efficiently as we increase the number of processors used.

Chapter 5

Optimising the distributed algorithm

Following on from the previous chapter, we increase the scalability and performance of the implementation of our algorithm. First, we develop a scalable approach to constructing large simulations. Subsequently, we apply a number of iterative improvements to the implementation that reduce the computational cost of simulating a specific test problem from over two hours to under two minutes. Finally, we develop a simple load-balancing algorithm and show that it reduces the inherent balance problem for our space decomposition, and consequently improves the parallel performance for spatially heterogeneous (or non-uniform) distributions of cells.

5.1 Introduction

In the previous chapter we presented a novel parallel algorithm for the off-lattice, individual-based model described in Chapter 3. We noted, however, that our aim of simulating large-scale populations of cells and achieving significant speed-up is restricted by computational inefficiencies.

In particular, the scaling of our basic algorithm suffers from three problems: first, for a fixed initial problem size the memory footprint grows as we increase the number of processors – reducing the size of problem we may solve in parallel. Second, the large amount

of data generated from serialising halo cells into messages means that the cost of communication is high, which limits the speed-up we attain when using multiple processors. Finally, we noted that our decomposition of physical space is blind to the actual distribution of cells within the domain. This leads to a poor balance of load, particularly for growing populations of cells such as monolayers and spheroids, where cells are unevenly distributed between processors.

We address each of these issues in turn in this chapter. In Section 5.2 we develop an alternative approach to constructing the simulation for large initial numbers of cells (the initial condition) which we show to be highly scalable. The work in Section 5.3 reduces the cost of local computations of cell neighbour pairs by using a container that is optimised for the most intensive part of the computation. In Section 5.4 we introduce two benchmark parallel scaling tests to guide the development of parallel optimisations. In Section 5.5 we make a restrictive assumption on the underlying architecture that enables us to shrink the size of the messages exchanged between processors. In Sections 5.6-5.7 we investigate overlapping communication and computation to limit the cost of transmitting large messages and reduce the inefficiency of blocking communication functions. Finally, in Section 5.8 we address load-balancing for spatially non-uniform cell distributions by developing a simple dynamic load-balance algorithm.

5.2 Efficient construction

We noted in the previous chapter that a principal inefficiency of the parallel algorithm is that, for a fixed problem size, its memory footprint grows as we increase the number of processors. In order to construct and then decompose the physical domain, each processor must know the initial location of each cell. We achieve this by having each processor store an initial copy of all cells, and then discard those cells whose location is outside the local portion of the physical domain. This approach means that we do not change the interface for the user: all initial cells are specified in the simulation program and hence constructed

on all processors. However, using this approach, the size of problem that we may solve is reduced as each processor requires enough memory for the entire initial condition.

As an example, in a distributed-memory architecture, the problem size, in terms of memory footprint, is limited to the size of each node's main memory divided by the number of processors per node. For the distributed architecture used here, which is composed of a number of 8-processor nodes, each node having a single 16Gb memory store, the theoretical upper limit of the problem size is 2Gb. This equates to approximately 2×10^6 cells in a 2D monolayer simulation. In practice, the limit is smaller due to memory required for other data structures in the algorithm, including communication buffers, as well as demands placed on the computer by other tasks.

By observing that, in decomposing the physical domain, it is only necessary to obtain an axis-aligned bounding box around the initial cell centres, we can overcome this problem. Writing the initial location of each cell in a simple text archive in the hard-disk memory, it is possible to calculate the bounding box by successively loading each cell location and comparing the co-ordinates with the maximum and minimum of all previously loaded cells. Once this is complete we use the bounding box to create and decompose the physical domain. Subsequently, we reload the archive file one cell at a time and each cell is added to the local memory only if its location is in the local portion of the physical domain. Figure 5.1 outlines this algorithm.

We confirm that this approach solves the problem by measuring the memory footprint for a fixed-size problem on an increasing number of processors of a distributed-memory cluster.

We record Resident Set Size (RSS) for each process (with each process assigned to a single physical processor), which is the active part of RAM for that process. We choose RSS as a measure of memory consumption since it is representative of total memory use and is easily accessed during the execution of a program. We obtain the data through system calls after the construction of the simulation. Listing 5.1 shows the system calls used.

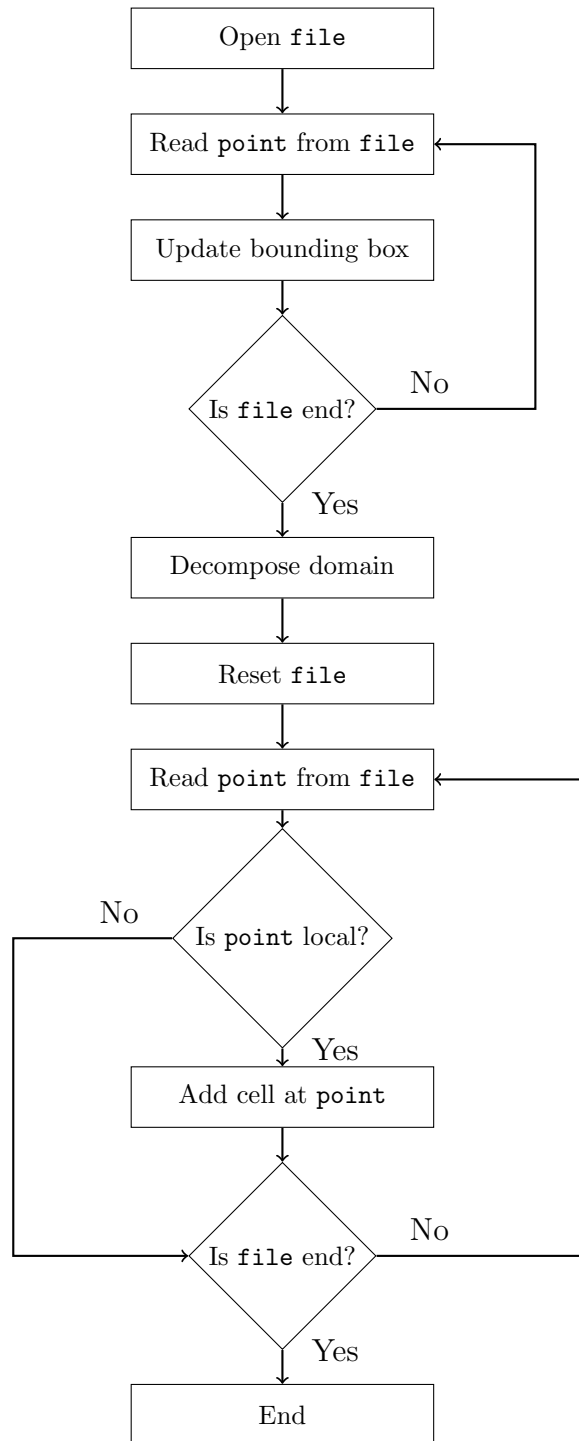


Figure 5.1: Algorithm for efficient construction of a simulation with a large initial condition from file.

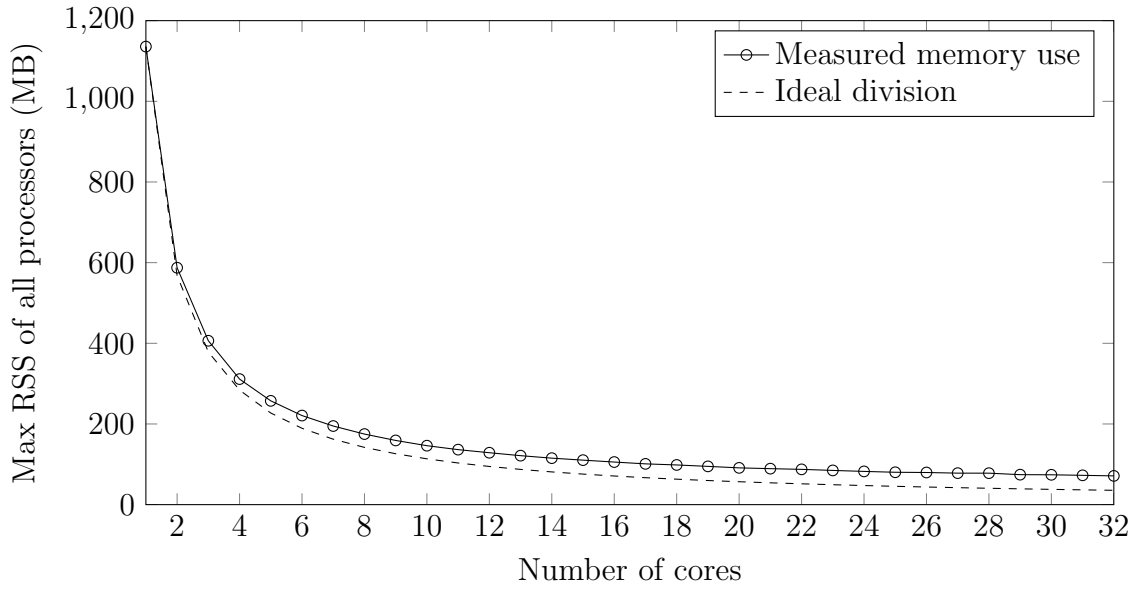


Figure 5.2: Memory use, measured using maximum RSS over all processors, using efficient construction algorithm for a simulation of 1,024,000 cells in a 2D monolayer.

To test the scaling of this approach we construct a fixed-size simulation of 1,024,000 cells evenly distributed throughout a 2D physical domain and consequently evenly distributed between processors.

```

1 void PrintMemoryUsage()
2 {
3     struct rusage rusage;
4     getrusage(RUSAGE_SELF, &rusage);
5
6     // Convert KB to MB
7     double max_res = (double)(rusage.ru_maxrss)/(1024);
8
9     std::cout << max_res << std::endl;
10 }

```

Listing 5.1: C++ system calls used to obtain RSS of the current process.

Figure 5.2 shows the maximum RSS recorded across all processors after construction of the test simulation for between 1 and 32 processors of a distributed-memory cluster. The recorded memory use closely tracks the idealised division of memory, given by the memory use recorded on one processor divided by the current number of processors. There is a slight overshoot of this idealised division, due to constant memory costs of the simulations, such as local data structures and objects used to evolve the simulation.

By using this construction algorithm, provided cells are evenly distributed through the spatial domain, the problem size is limited only by the total memory available in the computer independent of the number of processors used. For the distributed-memory architecture used in this project, the previous limit was a simulation with up to a 2Gb memory footprint. Using the efficient memory algorithm we may, in principle, simulate a problem with a memory footprint of 1Tb. While a 1Tb simulation would be possible on the shared-memory architecture with the previous approach, we are forced to use a single processor. With the new construction algorithm, given an even spatial distribution of cells, we can in principle use all available processors for large simulations. Extrapolating from our previous measurement, 1Tb of memory would enable approximately 10^9 cells in a 2D monolayer. Such a large simulation is, however, not possible due to the prohibitive computational cost of calculating neighbour pairs and cell forces for this many cells. Consequently, we turn our attention to optimising the use of processing power in our parallel algorithm. In the next section we report a computational inefficiency found in the implementation of the algorithm for calculating the neighbour list in Chaste. In the following sections we report methods for improving the parallel scaling of our algorithm for a given problem.

5.3 Efficient containers

To construct a list of cell neighbours we dynamically insert pairs of pointers to cell objects into a container. This container is then used to update forces between cells. For such containers, Chaste uses the Standard Template Library (STL), which offers a number of choices of containers with different features. In Table 5.1 we outline the three types of extensible containers that are suitable for the neighbour list, along with their key features. We pay particular attention to the complexity (in terms of operations) of inserting elements into the container, since this is the principal operation when dynamically constructing the neighbour list.

Name	Properties	Insertion complexity
vector	Variable size array. Uses contiguous memory.	Worst: $O(n)$. Best: $O(1)$.
list	Doubly-linked list. Non-contiguous memory.	Worst: $O(1)$. Best: $O(1)$.
set	Unique elements. Elements stored with weak ordering.	Worst: $O(\log(n))$. Best: $O(1)$.

Table 5.1: Types of extensible container in the STL library. Operational complexity of insertion is in terms of the size, n , of the container.

The vector class implements C-style arrays, which are able to dynamically vary their size. The most efficient form of insertion for vectors is a ‘push-back’, which adds a single element to the end of the container. The complexity of this operation depends on the amount of additional memory allocated by the vector prior to the ‘push-back’, but is at best constant and at worst linear in the number of elements in the vector.

The list class implements a doubly-linked list, where elements in the container are not stored contiguously, but maintain the order of the container by storing pointers to their predecessor and successor. By not enforcing contiguous memory, a list guarantees constant complexity for insertion, since only the pointers of at most two elements in the container are altered.

The final container to consider, which is used in the original Chaste implementation, is a set. The key features of a set are that it stores elements uniquely, and in a strict weak order defined by a given comparison operator for the type of data stored. Since sets are implemented as a binary search tree to maintain these properties, insertion has best-case constant complexity and worst-case $O(\log(n))$ for the size, n , of the container.

It should be noted that the new C++ standard, C++ 11 [C++11, 2011], offers additional containers such as unordered-sets, which do not enforce the ordering property of standard sets and therefore perform faster on insertion. However, we do not consider these containers because C++ 11 is not supported by older compilers used by the wider Chaste community.

While the complexity analysis may suggest that, for an algorithm that intensively

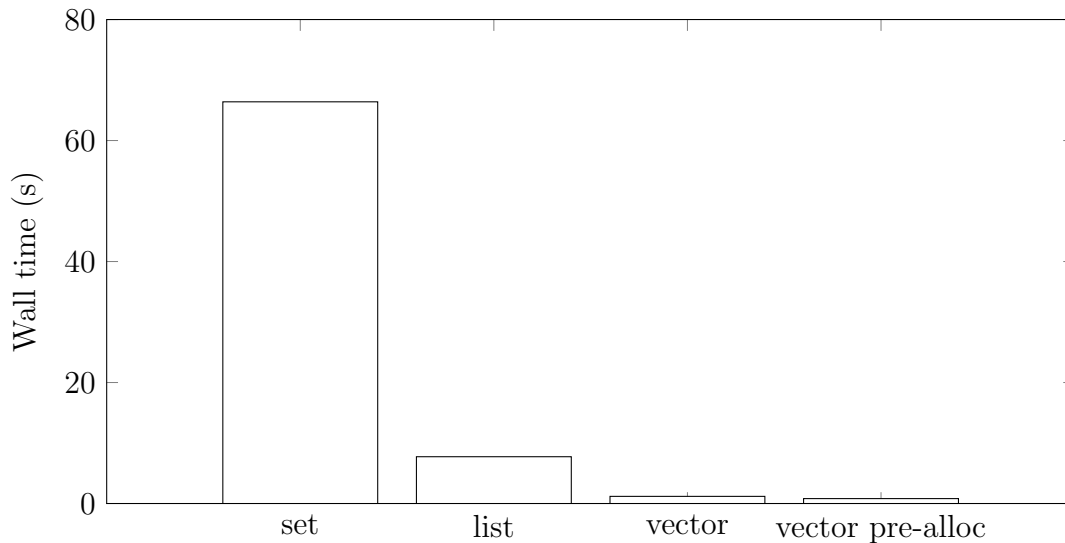


Figure 5.3: Performance of 10^7 repeated insertions / push-backs for different types of container in the STL library.

inserts into a container, a list may be the optimal choice, we show that, in practice, this is not the case. While the complexity of the insertion operations gives the marginal cost of adding another element to a container, in constructing our neighbour list we grow a container from empty by repeatedly inserting new pairs of neighbouring cells. We are therefore concerned with the speed of insertion for small as well as large containers. By testing the wall time taken to insert (or push-back) 10^7 pointers into each type of container we see in Figure 5.3 that vector outperforms the other containers: by over 60 times, in the case of sets, and around 7 times in the case of lists. We also see that if the size of the vector is pre-allocated then we obtain a further performance increase, since no further memory re-allocations are executed. Hence, for our specific application, we expect to see a performance increase from replacing the set container with a vector or list.

The principal benefit of using a set is that it guarantees $O(\log(n))$ performance in a search for an element compared to $O(n)$ for a vector or list. However, once constructed, we only iterate through the neighbour list, and no searches are carried out. We therefore replace the use of a set with a vector in the Chaste implementation. Since the size of the neighbour list is unknown and variable, we cannot pre-allocate with any accurate bound on the size of the container. Given that the performance benefit of pre-allocation has

	Total wall time (s)	Neighbour list calculation (s)
set	8517.458	5078.653
vector	2270.480	345.592

Table 5.2: Performance of Chaste implementation using different containers for the neighbour list. Simulation of 100 time steps for 1,024,000 cells in 2D.

been shown to be minimal, we choose not to pre-allocate the vector. While a vector does not provide the guarantee of uniqueness of elements ensured by a set, this is ensured since each cell belongs to a unique cuboid so the pair (Cell1, Cell2) is only ever considered once. Furthermore we use the unique cell identifiers described in Section 4.3.4 and ensure we only insert pairs of cells where the first entry has a smaller identifier than the second to avoid double-counting pairs such as (Cell1, Cell2) and (Cell2, Cell1).

To demonstrate the benefit of using this more efficient container in the Chaste implementation, we test a 2D monolayer of 1,024,000 cells simulated for 100 time steps (so that one-off costs such as simulation setup are a negligible proportion of simulation time). Table 5.2 gives the observed wall time using sets and vectors for the neighbour list when simulating on a single 2.80Ghz Intel Xeon CPU.

We observe a 3.75 times speed-up for the overall simulation, and a 14.7 times speed-up for the calculation of the neighbour list which originally accounted for 59% of simulation time, and subsequently used 15% of simulation time. Compared to 60 times speed-up for inserting pointers, we only see an approximate 15 times speed-up for constructing the neighbour list because this operation also includes other tasks, such as iterating through the potential neighbours identified in the cell-list method, which are not improved by changing the neighbour list container.

5.4 Initial scaling results

Having dealt with memory efficiency in Section 5.2 and improved serial computation speed in Section 5.3, we now consider developments to the parallel scaling of our algorithm. By optimising the way we use computational resources in parallel we are able to increase the

speed-up for a fixed-size problem, as well as enabling better simulation time for a problem that grows with the number of processors. In this section we introduce some canonical tests of scalability and present the initial results for our basic parallel algorithm described in Chapter 4 along with the optimisations in the previous two sections.

Benchmark scaling tests

To provide a benchmark for the parallel scaling of our algorithm we use two test simulations. From the plethora of different biological applications that may be simulated using this model in Chaste it is difficult to choose a single representative test. However, as demonstrated in the previous chapter, simulations typically can be divided into two categories: those that exhibit large movement and growth in cell numbers, and those that do not. Since performance of the former category depends on the load-balance arising from the space decomposition, we postpone consideration of these simulations until the end of the chapter and use examples from the latter category as benchmarks.

As we observed in the introduction to the previous chapter, our development of a parallel algorithm has two primary goals: firstly to enable larger problems to be simulated for a given amount of computation time, and secondly to enable a simulation of a given size to run faster – either to make their computation time tractable for a user or to enable multiple runs of a single simulation.

To address the first aim we test the weak scaling of the algorithm. Weak scaling is the speed-up achieved in parallel when the problem size grows with the number of processors used. We use relative speed-up as a measure of weak scaling: suppose we test a problem of size N_p cells on p processors, which takes t_p seconds of wall time to simulate, then we define the *relative speed-up* for p processors as

$$\frac{t_1/N_1}{t_p/N_p}.$$

The benchmark we consider for weak scaling is a rectangular 2D monolayer of cells

with no cell division. Drawn from the application in the next chapter where cells are constructed of a collection of points in space, rather than a single cell centre, we create a repeating pattern of 1024 cells which have been placed in a circular domain as shown in Figure 5.4. These circular domains are repeated in rows of 10 so that each processor owns a 10×10 region of circular domains, containing a total of 102,400 cells. To ensure that an equal number of cells are assigned to each processor, an additional 10 rows of circular domains are added for each additional processor, as shown in Figure 5.4(b). Consequently, the problem size on p processors is $102,400p$ and the relative speed-up is given by

$$p \frac{t_1}{t_p}.$$

To address the second aim, we test strong scaling, which is the improvement in performance for a fixed problem size as the number of processors is increased. To quantify strong scaling, as in the previous chapter, we use speed-up (t_1/t_p) as the metric.

The simulation we consider for strong scaling is 100 rows of 10 circles from the weak scaling construction, giving 1,024,000 cells. Both the weak and strong scaling experiments include no cell division and cells are initially separated by no less than 75% of their equilibrium distance so movements are small relative to the equilibrium distance. Each simulation is run for 100 time steps to ensure that one-off costs in setup and destruction do not dominate the measured computational cost.

In the discussion of our results we also refer to efficiency and relative efficiency, which are defined respectively as the speed-up divided by the number of processors used, and the relative speed-up divided by the number of processors used. This gives a metric for how much of the computation time is spent solving the problem as opposed to communicating and synchronising between processors.

We consider two HPC systems for testing our algorithm scalability, the key features of which are summarised in Table 5.3. For testing weak scaling we use a distributed MPI cluster. This system contains 512 2.8Ghz Intel Xeon processors arranged on 64 nodes, each with 8 processors. Each node has a 16Gb shared-memory and nodes are

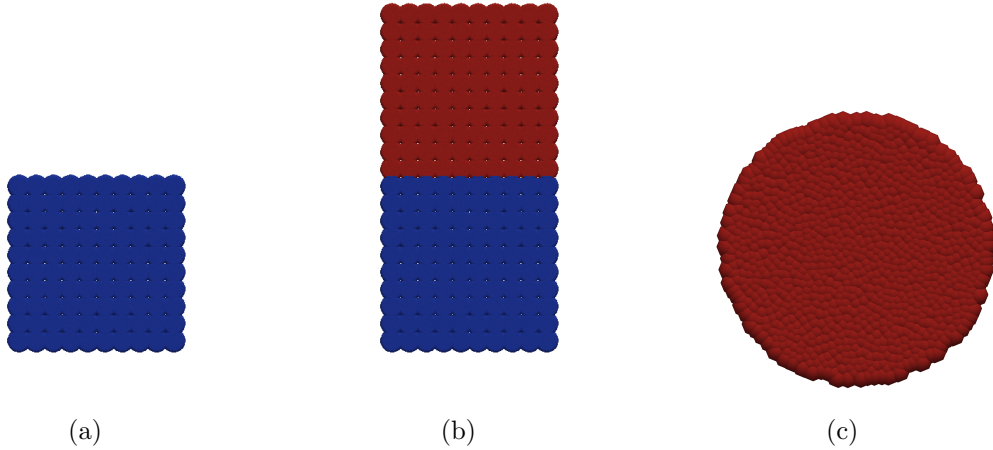


Figure 5.4: The initial condition for the weak scaling experiment on (a) one processor and (b) two processors. (c) Each macroscopic circle contains 1024 cells close to mechanical equilibrium.

	Processors	Nodes	Memory	Communication
SGI ICE 8200 MPI Cluster	512	64	1Tb (64x16 Gb)	Infiniband / shared-memory.
SGI UV 100 shared-memory	64	1	1Tb	shared-memory.

Table 5.3: Specifications for the two test HPC systems used to profile our parallel algorithm.

connected through a dual QDR infiniband network. For the strong scaling experiments we use a shared-memory system which has 64 2.67Ghz processors all connected to a single 1Tb memory store. Consequently all message passing on this system is executed through shared-memory, rather than a network. We choose these two different systems to demonstrate both the flexibility of our implementation for both shared- and distributed-memory machines, but also to highlight the impact of the latency issues associated with using distributed-memory where communication through a network suffers from significantly poorer performance. Both systems are provided and maintained by the Oxford Advanced Research Computing¹

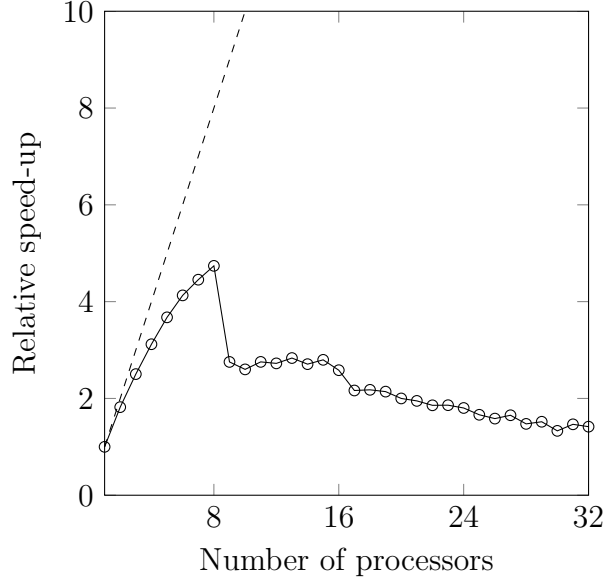
¹The author would like to acknowledge the use of the University of Oxford Advanced Research Computing (ARC) facility in carrying out this work. <http://www.arc.ox.ac.uk/content/services>

Initial results

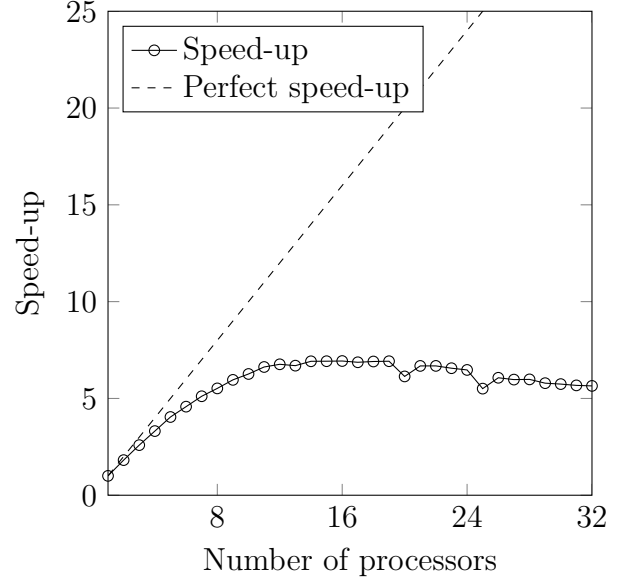
Figure 5.5 shows the results of our two benchmark scaling tests for the basic algorithm and implementation described in Chapter 4 along with the optimisations for memory and serial efficiency described in Sections 5.2-5.3. Figures 5.5(a) and 5.5(b) show the speed-up metrics as we increase the number of processors used, and Figures 5.5(c) and 5.5(d) show the corresponding proportion of simulation time for the key operations in the algorithm. ‘Migration’ is the cost of re-sizing the domain (described in Section 5.8) and reallocating cells between processors. ‘Halo cells’ is the cost of identifying and communicating halo cells between processors. ‘Cell neighbours’ is the cost of constructing the neighbour list and ‘Force’ is the cost of evaluating forces between pairs of cells identified in ‘Cell neighbours’. Values are averages over all processors. While other tasks such as output and memory re-allocation do consume CPU time, they are all smaller proportions ($< 10\%$) and do not grow as the number of processors is increased. For clarity we therefore omit the results of timing these operations.

In Figure 5.5(a) we see that the weak scaling simulation shows good relative speed-up for up to 8 processors on a single node with a relative efficiency of 90% on two processors and close to 60% for 8 processors. Once more than a single node is used, the higher latency associated with communication means that relative speed-up is reduced. Although we can solve larger problems on the distributed system, it takes substantially longer than a proportionally smaller problem on fewer processors. For example, there is 1.465 relative speed-up on 32 processors for a problem size 32 times larger. Consequently, parallel simulation time for the large problem is around 20 times longer than for the small problem in serial.

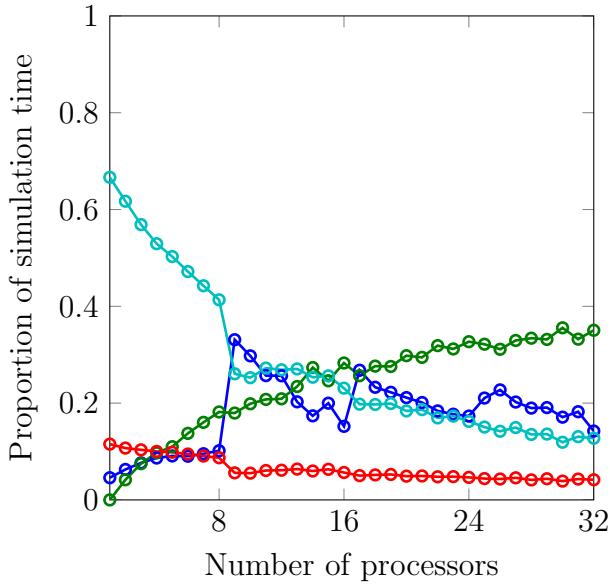
Figure 5.5(c) suggests that it is the growth of communication costs that causes this poor scaling. While this shows as an increase in ‘Migration’ costs, by looking at the variation in ‘Migration’ time across processors we can see this is likely to be due to the synchronisation of global communication in this method used to grow the domain size. The standard deviation of time spent in ‘Migration’ for 9 processors is 63.28 s compared



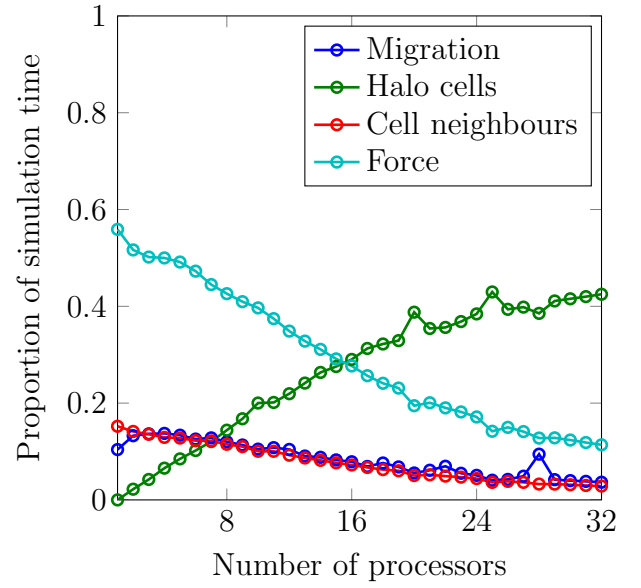
(a)



(b)



(c)



(d)

Figure 5.5: Baseline scaling for two test problems: (a) weak scaling on distributed-memory and (b) strong scaling on shared-memory. Proportion of simulation time spent in most costly methods for (c) weak scaling and (d) strong scaling.

to less than 2.5 s for fewer than 9 processors.

To further understand the causes of poor scaling on multiple nodes of our distributed system we test the communication cost between processors on the same and on different nodes of the system. By sending a fixed-size vector of 10^6 floating-point numbers using blocking MPI send and receive calls, and computing the average time taken to complete the send and receive operations, we observe a difference of over an order of magnitude in communications costs. Between processors of the same node, mean communication time is 0.1664 s with a standard deviation of 0.1142 and between processors on different nodes, communication time is 5.3573 s with a standard deviation of 0.1538. As a consequence exchange of halo and migrating cells between processors on separate nodes will be slower, and other processors will have to wait at synchronisation points. This can further be seen in Figure 5.6 where we present the total simulation time spent in the same categories as Figure 5.5(c) when the beginning and end of each section of the code which is timed has an MPI barrier inserted. This means that all processors enter and exit each section synchronously, reducing the observed effect of synchronisation on the timing results. We see here that the cost of calculating ‘Halo cells’, along with transferring migrated cells between processors, both scale poorly.

Figure 5.5(b) shows that strong scaling on the shared-memory system is also limited. Maximum speed-up of 8.2 is achieved on 16 processors with maximum efficiency of 91% achieved on 3 processors. Here, the substantial growth in the cost of halo cell communication can be seen in Figure 5.5(d), which accounts for the limited scaling of the problem.

These initial results suggest that the primary development required to address our aims is to reduce the cost of exchanging halo cells. This includes the cost of serialising and deserialising cells and of transmitting the resulting messages.

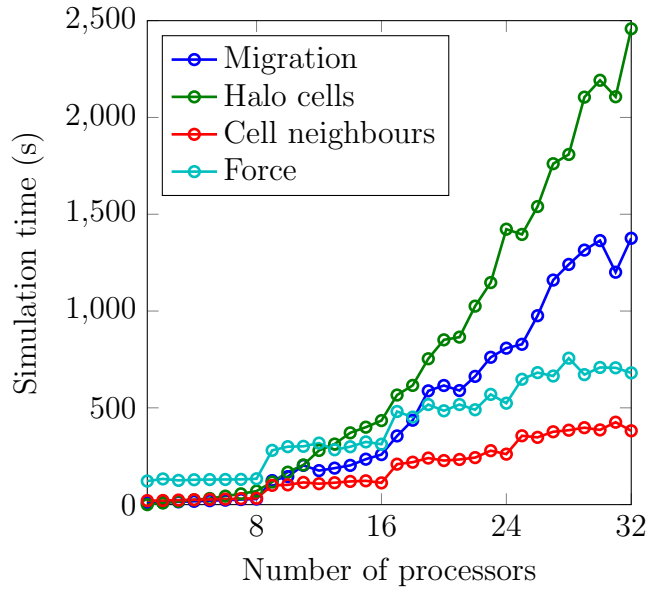


Figure 5.6: Total simulation time in each of the most costly methods for the weak scaling simulation with parallel synchronisation barriers at the start and end of each method. Note the problem size grows linearly with the number of processors.

5.5 Binary archiving

One of the novel features of the implementation of our algorithm is that any type of model cell in Chaste may be communicated, and hence simulated in parallel. We achieve this by using the ‘Boost Serialization’ library to convert instances of cell objects into strings of characters that can be sent between processors using MPI operations. However, as noted in the introduction to this chapter, the high cost of communication limits the parallel scalability of our implementation. To reduce the communication cost, we wish to decrease the size of the messages sent, the time taken to create them (by serialising cell objects), and the time taken to decode them (by deserialising the message into cell objects).

We achieve this by using the option to write Boost archives as binary strings rather than text strings. This means that archives are serialised and deserialised directly from memory, rather than being processed into ‘human-readable’ strings. Using this functionality places restrictions on the type of architecture that may be used to run the simulation code.

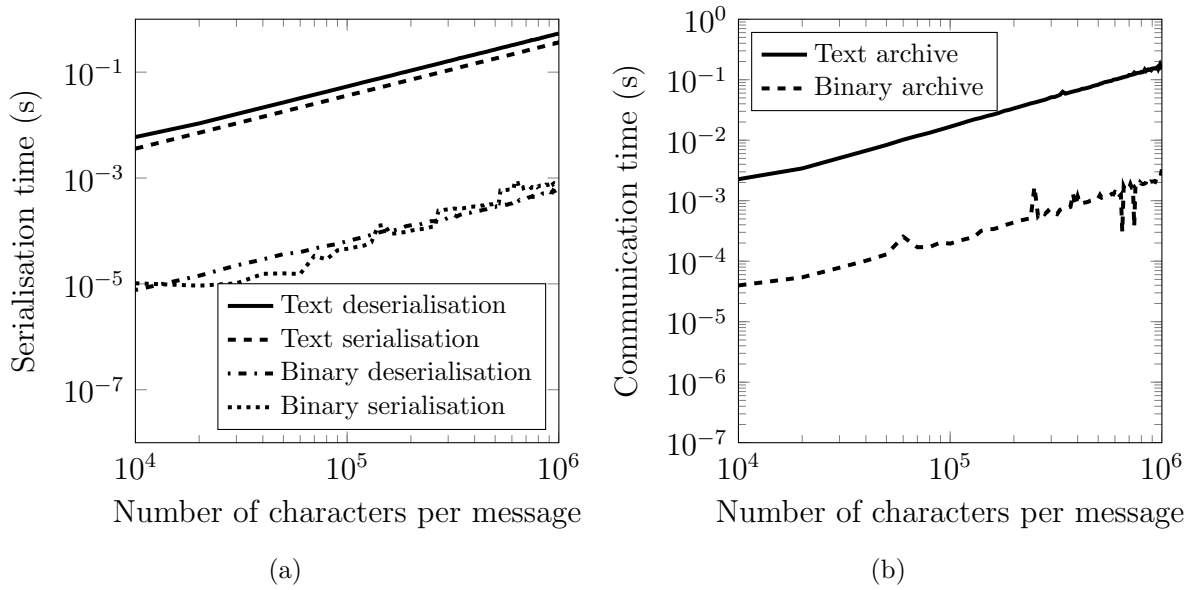


Figure 5.7: (a) Serialisation speed of text and binary archiving of an STL vector containing arbitrary characters. 100 different vector sizes were used ranging linearly from 10^4 to 10^6 components. (b) The communication cost associated with a communication of the text and binary archives of an STL vector of characters of increasing length.

The benefit of writing to text archives is that they can be read and written by different architectures and platforms (e.g. 32-bit / 64-bit, Windows / Linux). By using binary archives we insist that simulations are carried out on homogeneous systems, i.e. shared-memory machines or distributed systems of identical nodes. This restriction is unlikely to curtail the use of our software, since most academic HPC resources (to the extent of our experience) are homogeneous systems.

To demonstrate the benefit of using binary serialisation for communication messages, we first consider an abstract problem – serialising a contiguous string of characters. We measure the wall time taken to serialise and deserialise a vector containing between 10^4 and 10^6 arbitrary characters. By using blocking MPI communication functions we also consider the wall time to send the resultant archives through shared-memory. Figure 5.7(a) shows that serialising and deserialising with a binary archive is over two orders of magnitude faster than an equivalent text archive. Furthermore, we also see a reduction in the communication cost for sending the archive in Figure 5.7(b).

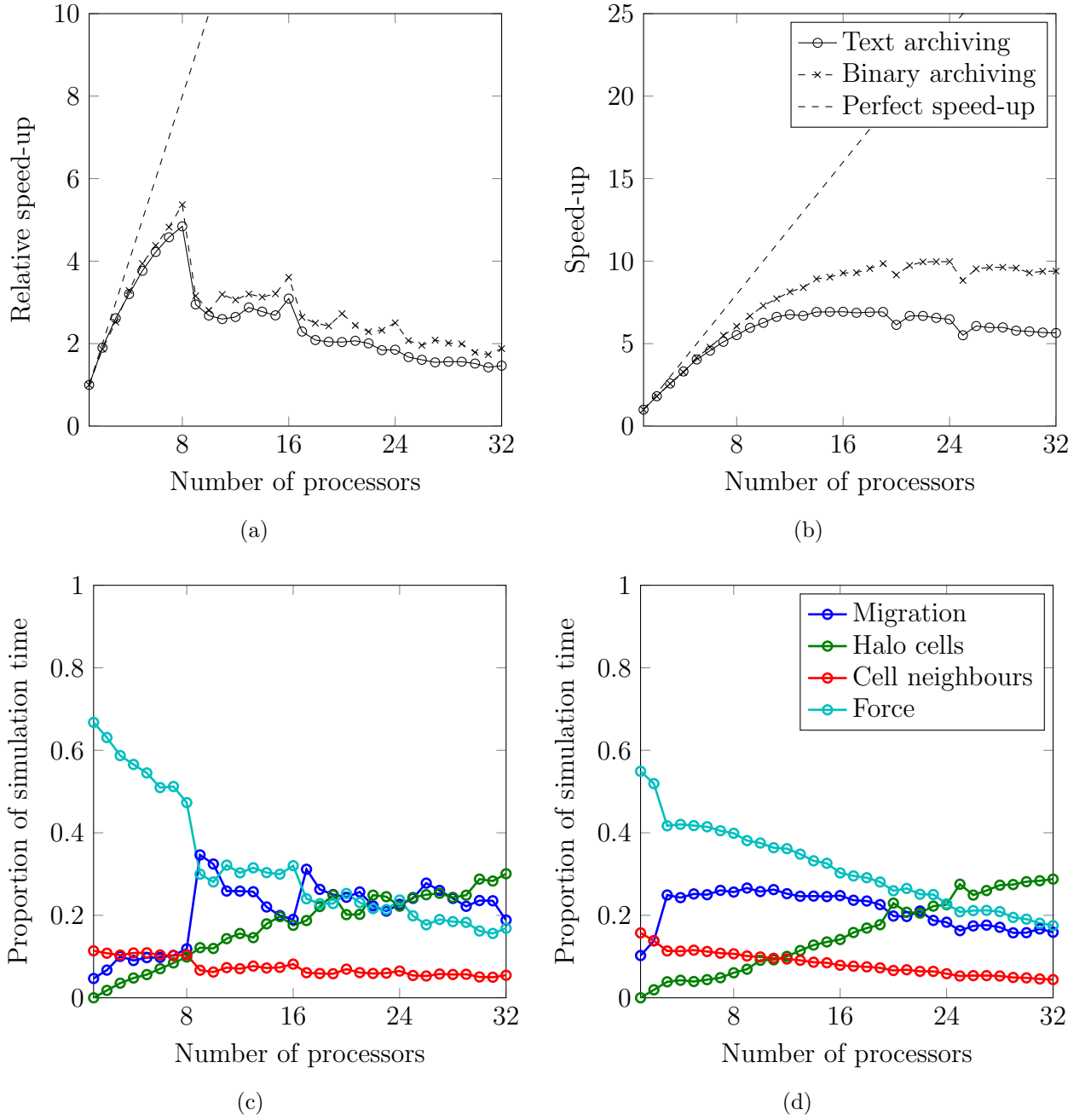


Figure 5.8: Scaling of test problems using binary archiving: (a) weak scaling on distributed-memory and (b) strong scaling on shared-memory of the parallel algorithm using binary archiving compared to text-based archiving. Proportion of simulation time in most costly methods for (c) weak scaling and (d) strong scaling.

To test the resulting improvement in the implementation of the cell population model we carry out the scaling experiments described in Section 5.4 for both text-based and binary archiving. Figure 5.8(a) shows weak scaling of the implementation using both the text and binary archiving modes. The simulation that uses binary archiving scales better as more processors are used and the problem size is increased. The performance increase is modest, however, compared to the increase in performance in the abstract problem. This may be understood in the context of the strong scaling experiment. In Figure 5.8(b) it can be seen that, for fewer than 8 processors, the benefit of binary archiving is negligible because communication costs are low. However as the number of processors is increased, the binary archiving mode substantially outperforms the text mode. However, on the distributed system, beyond 8 processors, the performance is dominated by the cost of transmitting messages between different nodes through the network, rather than serialising and deserialising cells. Although the total message size is also reduced, the number of messages is not reduced and the sent messages are still large enough so that communication dominates the total cost. Comparing Figure 5.5(d) to Figure 5.8(d) shows that the performance increase that we achieve on shared-memory is a result of reduced costs of transmitting halo cells, which only grow to dominate force calculation after approximately 20 processors, compared with around 16 for text archiving.

5.6 Non-blocking communication I

In the previous section we observed that, even when the message size is reduced through binary archiving of cell objects, the communication latency severely limits the parallel scaling of simulations. Since it is not possible to further shrink the message size whilst maintaining the extensibility and adaptability of our implementation, we investigate other communication optimisations.

By observing that, in most simulations, the halo region (i.e. the region whose cells must be sent to neighbour processors) is small compared to the whole local physical

domain, and hence much cell neighbour calculation is independent of the remote data, we may use non-blocking MPI functions to ‘hide’ some of the cost of communication. That is, by using send and receive functions that return before the communication is complete, we may carry out neighbour evaluations for cells that do not lie in the halo region while the send and receive operations are completed in the background. We split the description of this method into a brief outline of the relevant functions available in MPI in the next subsection, followed by an account of how these functions are used within our implementation. We then repeat the strong and weak scaling experiments to evaluate the benefit of this approach.

Non-blocking MPI operations

The basic MPI functions, `MPI_Send` and `MPI_Recv`, ensure that all memory buffers used in communication are ‘safe’ at all stages of a parallel program, independent of the exact order in which different processors call the functions. This is achieved by these functions returning control to the program (henceforth referred to simply as returning) only when the send or receive operation is complete. For example, the call `MPI_Send(&buf, ...)` only returns once `buf` may be safely edited without corrupting the send operation. Equivalently, `MPI_Recv` ensures that the receive operation completes, so that the receive buffer may be used, before returning.

While these methods absolve the developer of the burden of ensuring that all communications are complete before transmitted data are used, they can lead to parallel inefficiency. While a message is being transmitted the processor is unable to carry out other computation. As described by Gropp et al. [1999], the MPI standard offers advanced operations, known as non-blocking operations, to alleviate this problem. The two analogous non-blocking operations are `MPI_Isend` and `MPI_Irecv`. Both of these operations return as soon as the communication has begun. Consequently, the processor can do useful work but there is a danger that the send or receive buffer may be changed or used by subsequent operations before the communication is complete. If this occurs then

either the data being sent will be corrupt if the send buffer is changed, or the data used will be incorrect if the receive buffer is used. This is an example in parallel computing of what is commonly known as a ‘race condition’, that is the outcome of the program depends on the sequence or timing of uncontrollable events in the program.

To enable software that uses non-blocking operations to avoid race conditions, the non-blocking functions take an extra parameter over their blocking siblings. This parameter is a pointer to an instance of an `MPI_Request` object. This object may later be interrogated using the `MPI_Wait` function, which only returns once the communication operation of the request is complete. Through this separation of initiating and completing message transmission, we may overlap communication and computation. We show the prototype signatures for both blocking and non-blocking MPI functions in Listing 5.2.

```

1 MPI.Send(void *buf, int count, MPI_Datatype datatype,
2         int dest, int tag, MPI_Comm comm)
3
4 MPI.Recv(void *buf, int count, MPI_Datatype datatype,
5         int source, int tag, MPI_Comm comm, MPI_Status *status)
6
7 MPI.Isend(void *buf, int count, MPI_Datatype datatype,
8         int dest, int tag, MPI_Comm comm, MPI_Request *request)
9
10 MPI.Irecv(void *buf, int count, MPI_Datatype datatype,
11         int source, int tag, MPI_Comm comm, MPI_Request *request)

```

Listing 5.2: Signatures for blocking and non-blocking MPI functions.

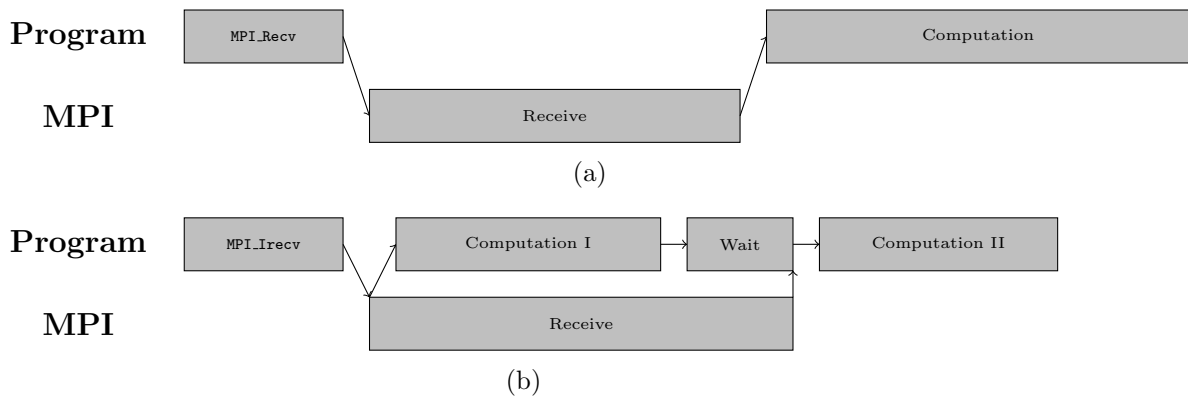


Figure 5.9: The flow of control in a parallel program using (a) blocking and (b) non-blocking receive operations. Using blocking communication control only returns from the MPI layer once the receive is complete. By returning before communication is complete, using non-blocking MPI, some computation (Computation I) that is independent of the data being received, may be carried out.

Supposing a part of a program has the structure of the blocking flow in Figure 5.9(a), by splitting the operations in ‘Computation’ into those that depend on remote data received (‘Computation II’) and those that do not (‘Computation I’), we may adjust the code to the structure of the non-blocking flow in Figure 5.9(b). Here the cost of receiving the message may be (partially) hidden by the computation independent of the message. The situation is analogous for `MPI_Isend` – we only need to ensure that the send buffer is unchanged until the send has been completed. To overlap communication and computation in our implementation, we identify parts of the algorithm with the structure of Figure 5.9(a).

Overlapping communication and computation in the cell population algorithm

The largest exchange of data in our algorithm occurs when halo cells are sent between processors so that the cell neighbour list may be constructed. Here we identify an opportunity to overlap the communication of halo cells with the construction of the neighbour list. We observe that, for a cell that is a distance greater than the interaction cut-off length, r_c , from the boundary of the local physical domain with a remote physical domain, all its neighbours must lie on the same processor as it. Therefore, the calculation of neighbours of such cells does not depend on data from halo cells. For clarity, this is illustrated in Figure 5.10. The neighbourhood of Cell 1 lies wholly in the region of processor 1 (C + D + E), and so all its neighbours are owned by processor 1. However, some of the neighbours of Cell 2 may lie on processor 2 in the halo region B, and therefore evaluation of the neighbours of Cell 2 depends on the exchange of halo cells.

The structure of the implementation of halo cell exchange and neighbour list construction using blocking and non-blocking operations is shown in Figures 5.11(a) and 5.11(b) respectively. In the blocking approach, after establishing which cells to send, we execute a send and receive using the `ObjectCommunicator` class with first the right and then left-hand processor (only one pair shown). We then use these data to calculate all cell neighbours once all communication is complete. To use non-blocking operations we split the calculation of the cell neighbours into two methods: the first calculates neighbours of cells away from the halo region and adds the pairs to a neighbour list. The second method iterates through all cells in the halo region and adds their neighbours to the same neighbour list. Consequently, a call to the two methods in sequence creates a complete list of all neighbours.

In order to use non-blocking operations in the `ObjectCommunicator`, we make a number of extensions. Having serialised the halo cell data into a binary string, we first exchange the string size with the neighbour processors so that memory for the receive operation may be allocated. This exchange of size must be blocking (as shown in Figure 5.11(b)), because the message size (parameter `count` in Listing 5.2) is required to post the receive

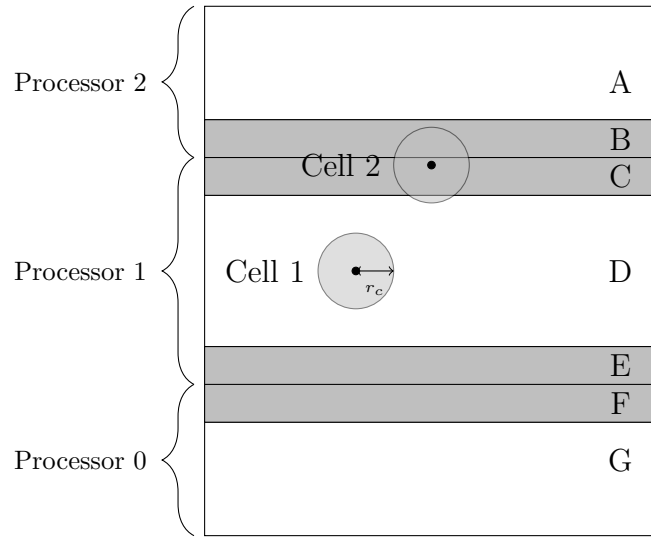


Figure 5.10: Neighbours of cells that lie outside of the halo regions (shaded grey), such as Cell 1, may be calculated independent of the halo cells. Cells near the halo regions of neighbouring processors, such as Cell 2, depend on halo cells and therefore can only be calculated when non-blocking communication has completed.

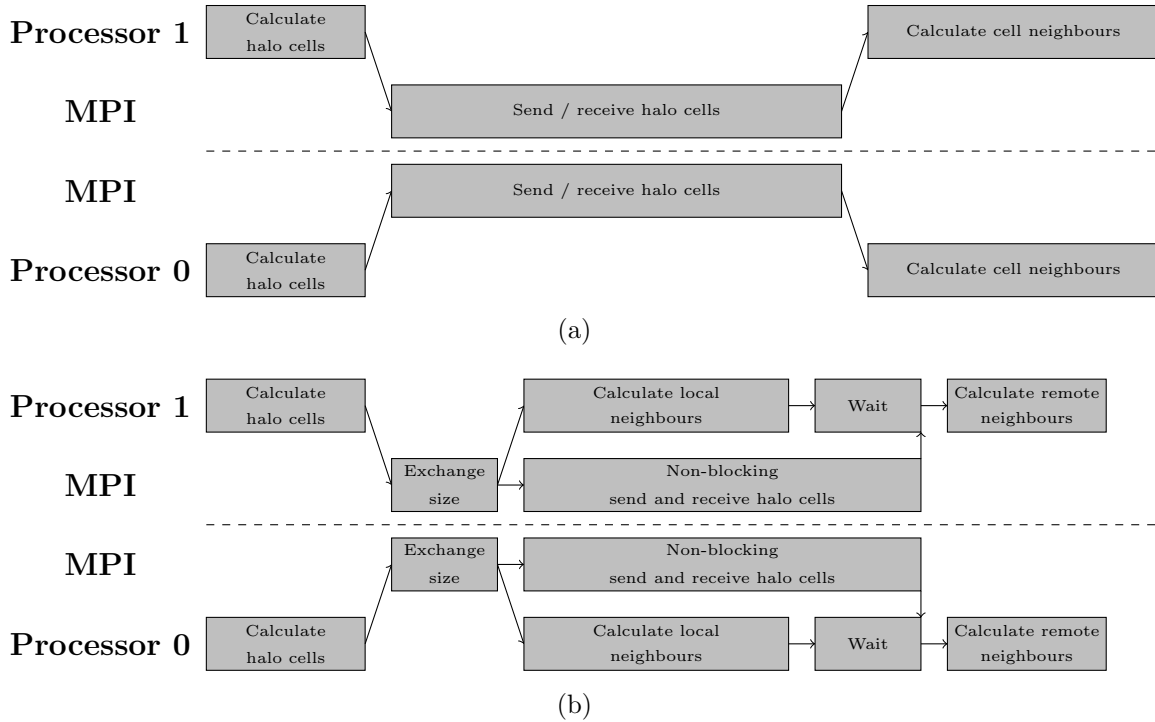


Figure 5.11: The flow of control in our algorithm when using (a) blocking and (b) non-blocking send and receive operations. We overlap the communication of halo cells and computation of local neighbours.

calls. Once this is complete, we post non-blocking calls for send and receive with the neighbour processors. Once the calculation of neighbours of local cells is complete we call a function in `ObjectCommunicator`, which carries out two operations: an `MPI.Wait()` to wait for the relevant non-blocking receive to complete, followed by deserialisation of the received binary string into halo cell objects. The neighbours of the boundary cells can then be computed.

By repeating the weak and strong scaling experiments we assess the improvement in parallel efficiency by using non-blocking send and receive operations. Figure 5.12(a) shows results for weak scaling of the test problem, where the number of cells grows with the number of processors. It can be seen that when all communication is through shared-memory (for fewer than 8 processors) little additional performance is gained. This follows from the relative high speed of shared-memory communication, where the message transmission is fast enough that we obtain almost perfect scaling. When some messages travel through the network connecting separate nodes, with higher latency (greater than 8 processors) we see a gain in parallel scaling from non-blocking communication. Since messages take longer to transmit, we gain performance by effectively having communication and computation operating in parallel. In the case of strong scaling on the shared-memory system, in Figure 5.12(b), we only see improvement in parallel performance where the blocking communication scaling begins to saturate, around 16 processors. By comparing Figures 5.12(c) and 5.12(d) it can be seen that the observed average cost of halo cell communication is reduced as a proportion of simulation time as more processors are used. This is because the functions return without completing the message transmission so the observed time sending and receiving messages is smaller. A limitation of this approach is the need to exchange the size of the message before the non-blocking communication calls can be made. As we discuss in the next section, this creates an implicit synchronisation between processors.

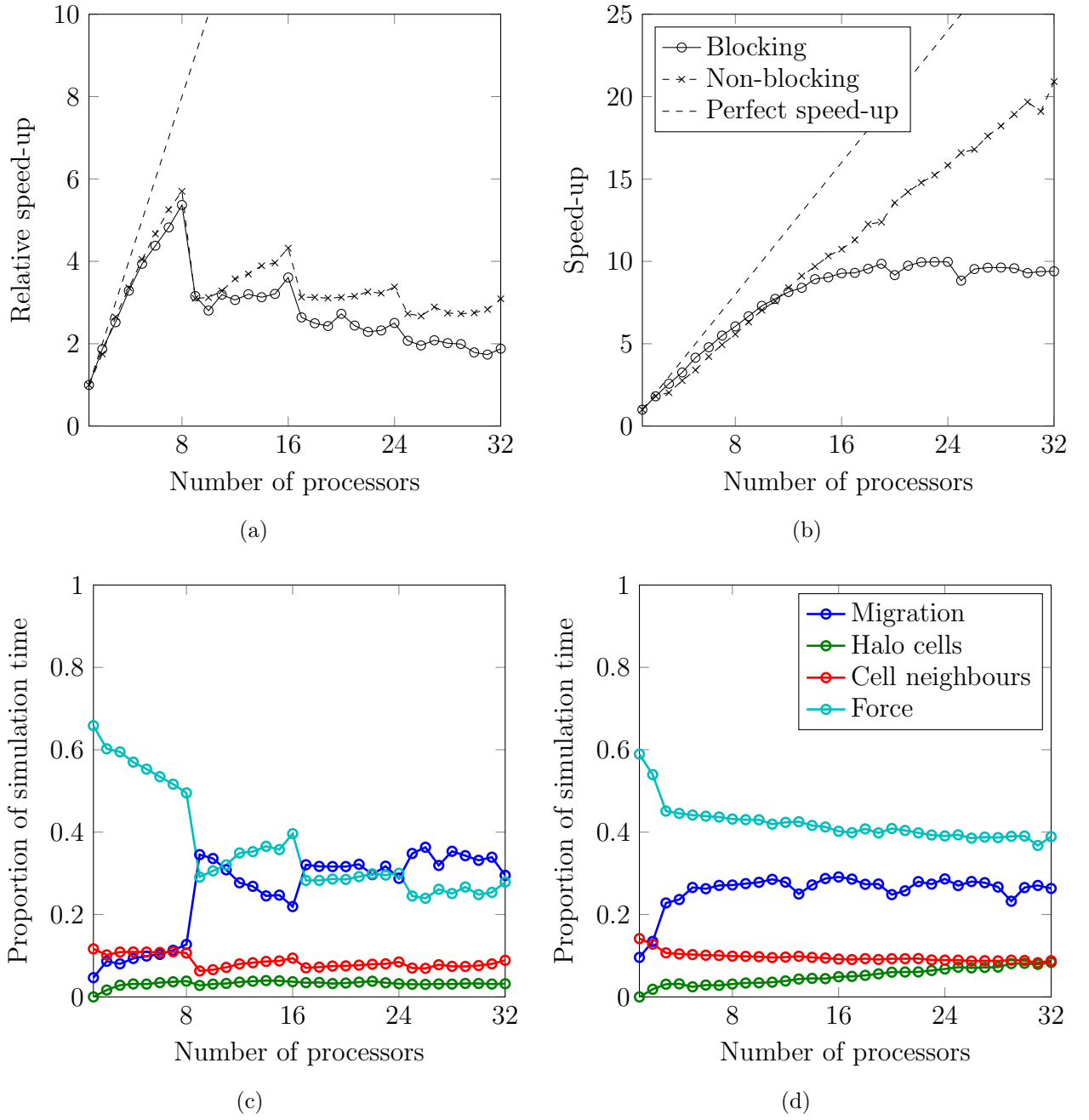


Figure 5.12: Scaling of test problems using non-blocking communication: (a) weak scaling on distributed-memory and (b) strong scaling on shared-memory of the parallel algorithm using non-blocking communication compared to blocking communication. Proportion of simulation time using non-blocking communication in most costly methods for (c) weak scaling and (d) strong scaling.

5.7 Non-blocking communication II

In addition to the cost of creating and passing messages between processors, a further cause of parallel inefficiency is synchronisation. A processor that depends on remote data for a calculation must wait for the processor owning the remote data to reach the stage of the parallel program where those data are sent. This creates a ‘synchronisation’ point and leads to a processor having to sit idle if the two processors do not reach the communication point at the same time.

Synchronisation can be a particular problem for our parallel algorithm where the exchange of halo cells and migrating cells between pairs of processors means that stepping through time in the algorithm must be tightly synchronised. Each processor must proceed through each time step with both of its neighbours.

In the previous section we showed how the delay caused by communication latency may be mitigated by overlapping communication and computation using non-blocking operations. This approach can also reduce synchronisation costs because a processor can post a non-blocking receive call and then proceed with local computation while the remote processor ‘catches up’. However, the algorithm in Section 5.6 for exchanging halo cells cannot take advantage of this break in synchronisation because there must be an exchange in the size of the message so that `MPI_Irecv` can be called with the correct parameters. Consequently, a processor may not proceed to local computations until these message sizes have been exchanged.

To break this synchronisation point we use the fact that `MPI_Irecv` can receive messages smaller than the parameter `size` passed to the function call without producing an error. We estimate a large upper bound for the size of the incoming message and, by ensuring that we use the same bound across all processors, we can exit the program with a warning if the size of the sent message is larger than the upper bound for receiving. If the error is thrown and the simulation exited, the bound can then be adjusted and the program re-run. Upon completion of the `MPI_Irecv` operation we use the `MPI_Get_count` operation to interrogate for the actual size of the message, before deserialising into halo cells. While

this avoids the synchronisation point, we must allocate a receive buffer which is the size of the upper bound for each communication. In many cases this is wasteful of memory since the true size of the message is much smaller, and consequently this approach will marginally reduce the size of problem that may be solved.

Figure 5.13(a) shows a modest improvement in the overall weak scaling achieved by removing the blocking exchange of halo cells, however the strong scaling (Figure 5.13(b)) shows virtually no change in the overall scaling, although we do observe a marginal decrease in total simulation time which can be attributed to a reduction in the pairwise synchronisation of processors when they exchange the size of their halo cell message. The weak scaling is still dominated by the cost of transferring data between different nodes of the test machine, while communication costs appear to be scaling well on the shared-memory machine.

Having improved the parallel scaling of our two test problems, we now turn our attention to the second class of simulation identified in the introduction: those that exhibit large movement and growth in cell numbers.

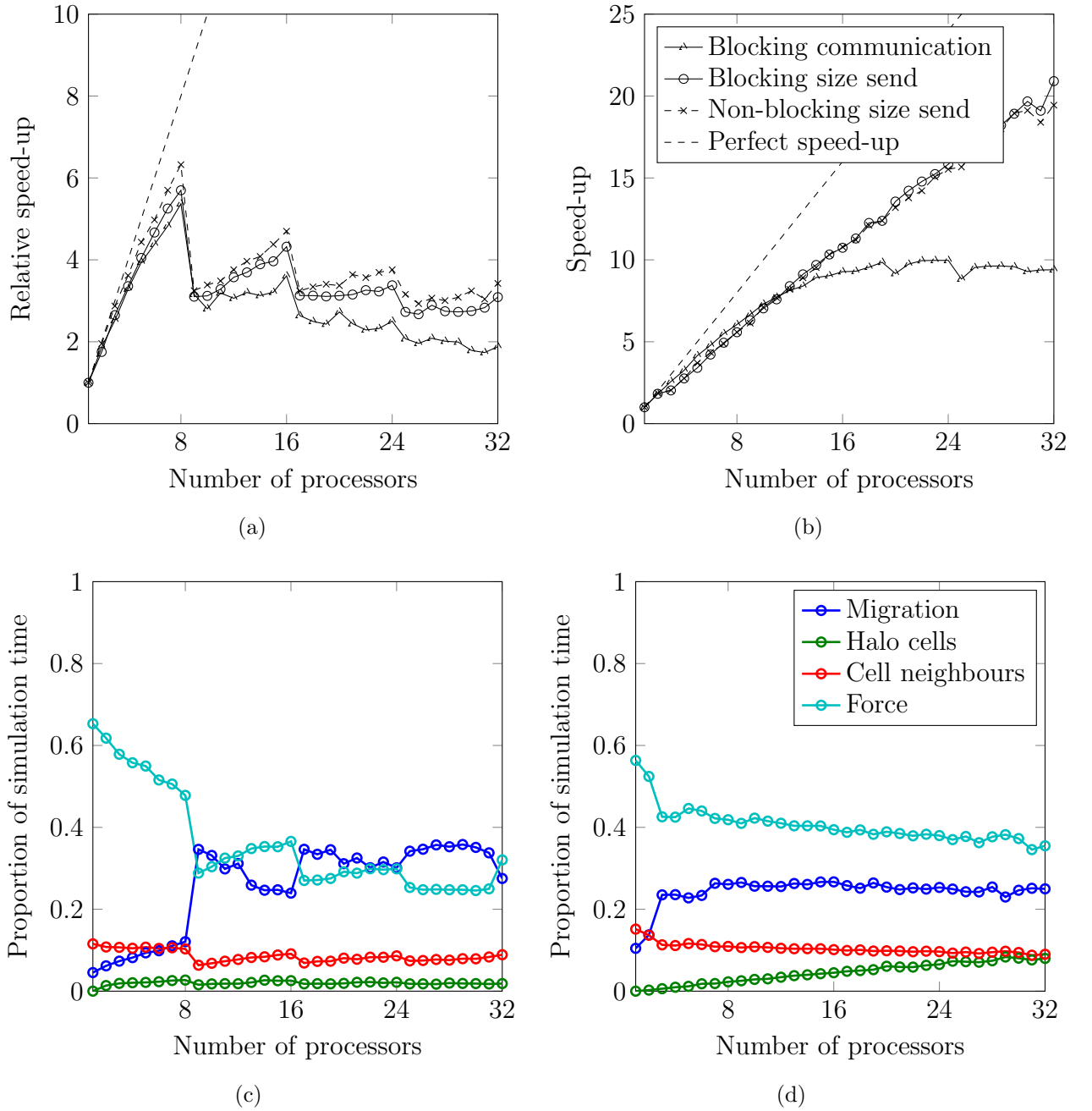


Figure 5.13: Scaling of test problems using non-blocking exchange of the message size (complete blocking included for reference): (a) weak scaling on distributed-memory and (b) strong scaling on shared-memory of the parallel algorithm. Proportion of simulation time using non-blocking communication of message size in most costly methods for (c) weak scaling and (d) strong scaling.

5.8 Load-balancing

As we noted in the previous chapter, particularly for simulations of growing cell populations, our spatial decomposition approach suffers from poor load-balancing because a static decomposition of space is used, and our approach to growing the space domain leads to a larger portion of space being assigned to the boundary processors. This impacts upon parallel performance because the simulation speed is limited by the processor that takes the greatest amount of time to calculate each time iteration. If one processor is overloaded, others will sit idle waiting for it to catch up. Load-imbalance is a ubiquitous problem in parallel computing and many parallel algorithms are designed specifically to avoid it. In cases such as ours, where naturally well-balanced algorithms, e.g. the AD and FD methods, are not favourable, further algorithms have been developed to balance the computational load. While an investigation of all possible methods is beyond the scope of this thesis, we consider the approaches available and construct a simple method that reduces load-imbalance and provides scope for future development.

Load-imbalance can arise either from a parallel computer with a heterogeneous architecture, where some processing components are more powerful than others, or from parallel algorithms that do not assign an equal amount of work to each processing unit in the system. The former problem is approached by static load-balance methods that balance the computational load in advance of run time [Tantawi and Towsley, 1985]. The latter problem, which we address, is approached by dynamic load-balancing at run-time.

Dynamic load-balance algorithms broadly implement versions of the diffusion approach: global load homogeneity is achieved by iteratively moving work from processors with a high load to processors with a low load [Cybenko, 1989, Corradi et al., 1999]. Such approaches are favourable because the exchange of load information between processors is local, i.e. only between small subsets of processors. Global approaches, where all processors share their local load data with all others and calculate a re-distribution of load, may achieve optimal load-balance without repeated iteration of the algorithm. However, the overhead of global communication between all processors means this approach is not scalable for

large systems and therefore poorly suited for most dynamic parallel applications [Corradi et al., 1999].

A prominent approach to dynamic load-balancing that falls into the diffusive category is the use of the Charm++ language [Kale and Krishnan, 1993]. Developed over the last 20 years, Charm++ is an extension of the C++ language that enables the definition of parallel objects called ‘chares’. Since these objects are inherently parallel they may easily be migrated between processors to balance computational load. Charm++’s daughter, AMPI [Huang et al., 2003], is an implementation of MPI that allows the creation of multiple virtual processes on each physical processor. These processes may then be automatically migrated between physical processors to correct load-imbalance.

In the specific field of spatially decomposed parallel algorithms, Merchant et al. [1998] have developed a dynamic approach to diffusive load-balancing spatially decomposed individual-based simulations. By dynamically shifting the boundaries between processors, as shown in Figure 5.14, they were able to achieve a doubling of the speed-up attained for an example of a communication-intensive simulation of schools of fish. The authors also consider allowing more freedom in the decomposition of the domain and subsequent movement of the boundaries which achieves further speed increases. Recently Fattebert et al. [2012] developed an extension of this approach for MD simulations, based on a decomposition of space into Voronoi regions. A small number of points (compared to the total number of molecules) are allowed to move through space to minimise the difference in load between their corresponding Voronoi regions. Based on these methods for load-balancing with migratable objects and space decomposition we propose a heuristic load-balance algorithm for our implementation, and show that while it does not guarantee an optimal distribution of load it significantly improves the performance of our algorithm.

We assume a homogeneous underlying architecture where each processing component is equally powerful and has access to equal local memory. Consequently, we develop a dynamic load-balancing approach based on the work of Merchant and the AMPI system of migratable objects, although we do not convert Chaste to Charm++ code. Using our

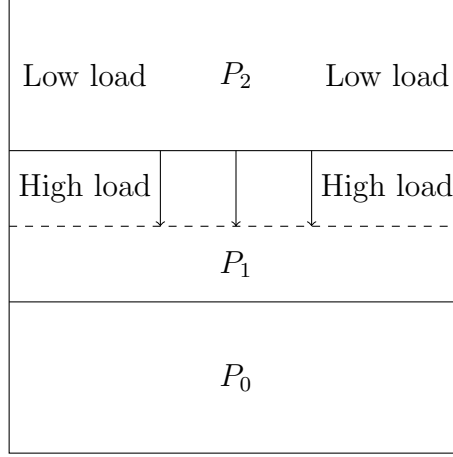


Figure 5.14: The load-balance approach of Merchant et al. [1998] for a spatially decomposed individual-based simulation. Load-balance is achieved by shifting the boundaries of processor domains to minimise local imbalance between adjacent pairs of processors. In this case the boundary is shifted so that P_2 , a lower loaded processor, takes some of the load from P_1 .

space decomposition approach, the minimal allocation of work to a single processor is one plane of cuboids, and so, by locally migrating planes of cuboids between processors (or equivalently shifting the boundaries in the space decomposition), we attempt to reduce global load-imbalance.

To calculate the difference in load between processors we first suppose that the amount of computation to be carried out by each processor may be assumed to be proportional to the number of cells that lie on that processor. This is a natural first step, and we discuss possible extensions in Chapter 7. Consequently, we define the load on a processor at a given time step to be the number of local cells.

To decide whether, and in which direction, to move each boundary the algorithm uses a more fine-grained description of the load distribution on each processor. Each processor computes the number of cells lying in each plane of cuboids in its local domain, obtaining a vector

$$\mathbf{W}^p = \left(w_0^p, \dots, w_{r_p-1}^p \right),$$

where p is the processor rank, r_p is the number of planes of cuboids in the local domain and w_i^p is the number of cells in the i^{th} plane.

Having obtained all local load information, the algorithm must decide whether to move each local domain boundary, and in which direction. Since moving a shared boundary must be a unanimous decision between the two processors concerned, the decision is made on one processor and transmitted to the other. In this respect each processor controls movement of its boundary with the processor with rank one less than its own.

Suppose we have neighbouring processors with ranks p and $q = (p + 1)$, then processor p sends its local load distribution $\mathbf{W}^p$ to processor q . Processor q then calculates the following quantities: current load-imbalance

$$\sigma_{pq} = \left(\sum_{i=0}^{r_p-1} w_i^p - \sum_{i=0}^{r_q-1} w_i^q \right)^2,$$

load-imbalance after receiving a strip of cuboids from p

$$\sigma_{pq}^{\text{recv}} = \left(\sum_{i=0}^{r_p-2} w_i^p - \sum_{i=0}^{r_q-1} w_i^q - w_{r_p-1}^p \right)^2,$$

and load-imbalance after sending a strip of cuboids to p

$$\sigma_{pq}^{\text{send}} = \left(\sum_{i=0}^{r_p-1} w_i^p + w_0^q - \sum_{i=1}^{r_q-1} w_i^q \right)^2.$$

By considering the resulting changes in balance

$$\delta_{\text{recv}} = \sigma_{pq}^{\text{recv}} - \sigma_{pq},$$

and

$$\delta_{\text{send}} = \sigma_{pq}^{\text{send}} - \sigma_{pq},$$

the processor decides to send if $\delta_{\text{send}} \leq 0$ and to receive if $\delta_{\text{recv}} \leq 0$. In the case that both $\delta_{\text{send}} \leq 0$ and $\delta_{\text{recv}} \leq 0$ we choose to send if $\delta_{\text{send}} < \delta_{\text{recv}}$ and receive otherwise. We choose weak inequalities to alleviate global imbalance problems by exploring more of the possible configurations of the space decomposition. A single iteration of this load-balance

algorithm is carried out periodically, with period, in multiples of the simulation time step, set by the user.

To test the efficacy of our load-balance algorithm we construct a simulation of a growing 2D monolayer cell population similar to that shown in Chapter 4 which exhibits poor load-balance as a result of the static decomposition of space. The simulation begins with an initial seed of 25 cells which are assigned a random birth time and divide subsequently every 12 hours, growing the population to approximately 10^4 cells after 100 hours. The simulation time step is set to $\Delta t = 3$ s. We consider two cases: where the load-balance algorithm is not applied, and where we re-balance every hour of simulation time. By testing on 8 processors of a single node of the distributed-memory machine (where all communication is through shared-memory) we obtain results comparable to Figure 4.11(b) from Chapter 4.

We find the algorithm is effective in reducing overall imbalance that is caused both by the process of growing the domain and by the static decomposition of space, demonstrated in Figure 5.15(b). At the end of the simulation total imbalance (the standard deviation of cell numbers across all processors) is reduced from 2069.9 to 219.3, which can be seen in the visualisations in Figure 5.15(c)-5.15(d). Figure 5.15(a) shows that, using the load-balance algorithm, we obtain three times speed-up on eight processors for the final hour of simulation time, compared to virtually no speed-up without load-balancing.

The load-balancing frequency, which must be set by the user, defines how often the balance operation is carried out. In Figure 5.16 we plot the total simulation time as a function of the load-balancing frequency. This shows that the optimal re-balancing frequency is approximately every 10^3 time steps, which is equivalent to approximately one hour of simulation time. This is in line with the approximate timescale of cell growth and division, which drives the large-scale motion of cells in this simulation and hence leads to load-imbalance. Figure 5.16 further suggests that cost of not load-balancing is greater than that of load-balancing too often, and in problems which have a less regularly defined growth and movement regime, it would be preferable to choose to re-balance regularly.

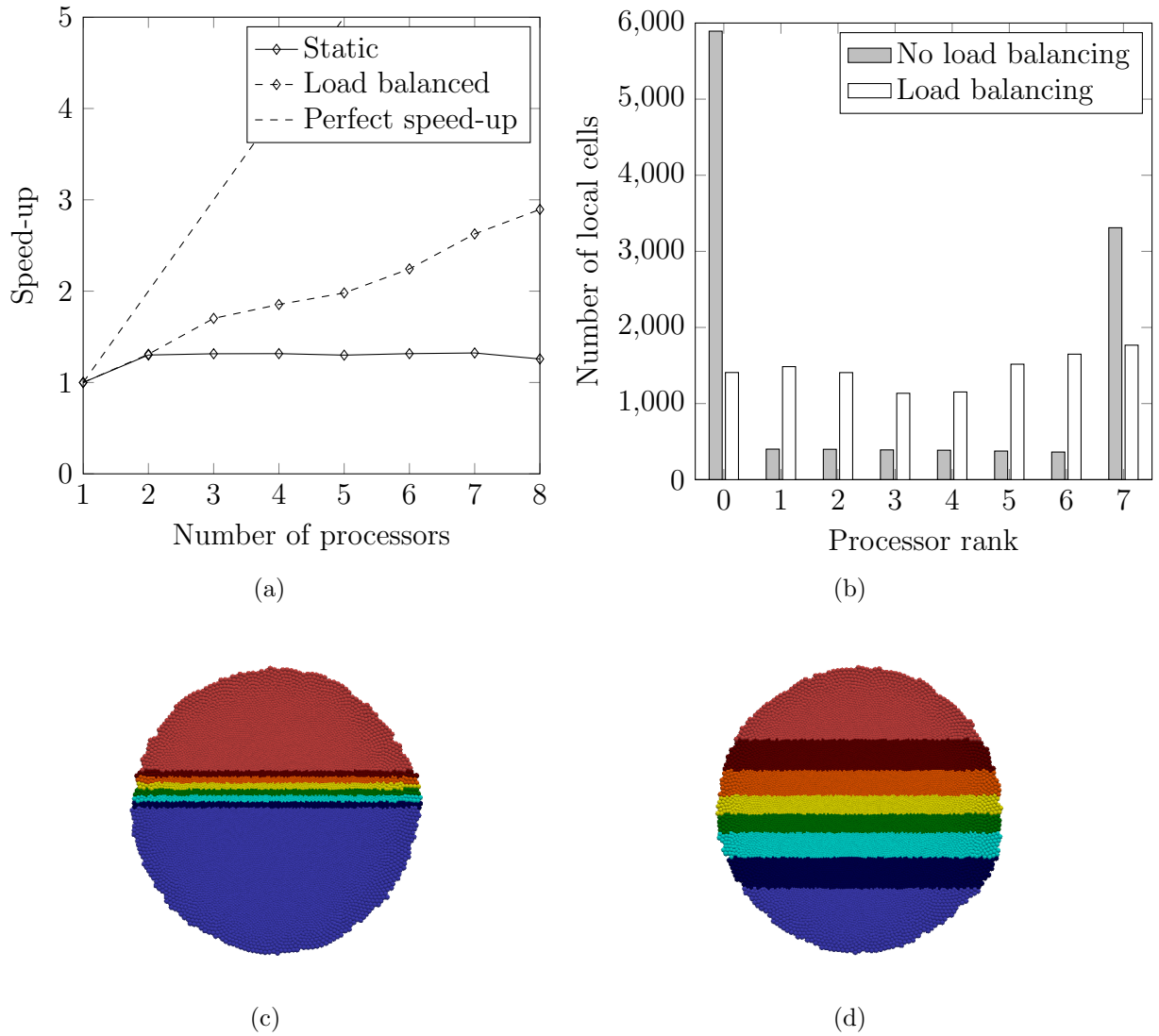


Figure 5.15: (a) Increase in speed-up using the load-balance scheme, with a load-balance iteration every hour of a 100 hour simulation. (b) The variation in cell numbers across processors for the final time step. (c) & (d) Visualisation of the final state of the cell population with cells coloured according to their owning processor.

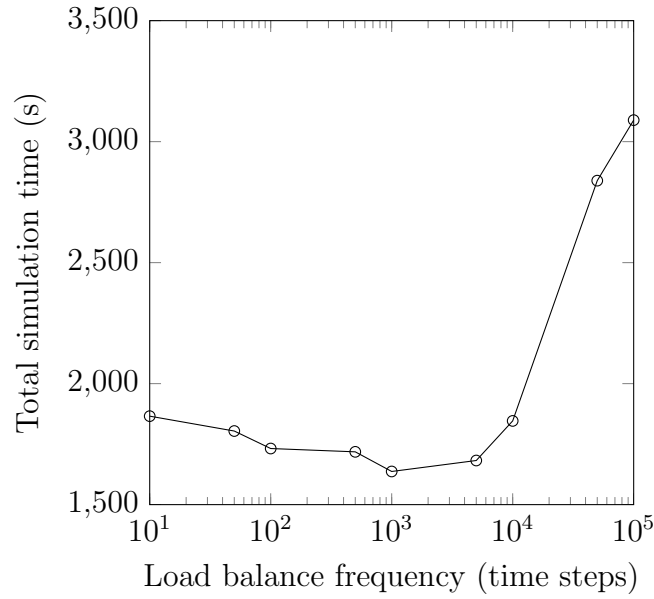


Figure 5.16: The total simulation time for the growing monolayer simulation as a function of the load-balance frequency.

Furthermore, the flatness of the minimum in this graph suggests that there is little to be gained from finely tuning the load-balance frequency.

5.9 Discussion

The purpose of this chapter is to further develop the algorithm described in Chapter 4 to meet our twofold aims of enabling larger as well as faster simulations of the off-lattice cell population model described in Chapter 3.

We have advanced our first aim by developing a scalable construction algorithm for large initial conditions. By writing initial cell locations to file, we decompose the domain without loading the whole initial condition to each processor's memory. Consequently, as we increase the number of processors, the initial memory footprint stays approximately constant for well load-balanced problems. This is a significant advantage over replicated data methods such as AD and FD reviewed in Chapter 4 which consume large amounts of memory by storing all cell data on all processors. For poorly balanced problems, memory use will not be evenly distributed between processors, meaning that for large,

irregular problems the work on load-balancing in Section 5.8 improves memory as well as computation time scalability.

Although memory use scales well, even for well-balanced simulations computation time of the original parallel algorithm is dominated by the cost of communication as we increase the number of processors. This is particularly inefficient in a distributed architecture where we have shown that message transmission costs can be an order of magnitude higher between physically separated nodes than through shared-memory.

To reduce this inefficiency we made the restrictive assumption that any distributed system used by our algorithm is homogeneous, enabling us to use binary serialisation to create and decode messages. In initial external tests we saw that binary serialisation of simple data types is over two orders of magnitude faster, and creates proportionally shorter messages than text-archiving. However, while we observed a significant improvement in scaling on the shared-memory machine, where message construction cost is high relative to the cost of message transmission, the improvement on the distributed-memory system was only marginal. Although message creation is faster and the message shorter, the simulation time is still dominated by the cost of transmitting the resulting message. Some of this discrepancy in performance between our simulation and the external test may be attributed to the need to serialise dynamic type information for the cells (i.e. data of derived types decided at run time) rather than basic data types used in the external test. However if this were the only limitation we would expect similar changes in scalability for both shared- and distributed-memory systems.

While the method of communicating Chaste datatypes using serialisation creates messages that are costly to transmit through a distributed network, its benefits are that it maintains the extensibility of Chaste and requires small changes to the classes to parallelise the simulation compared to using specialised data communication. Therefore, we have investigated methods for mitigating the high communication cost by executing communication and computation independently.

By developing extensions of the `ObjectCommunicator` class that enable any Chaste

object to be communicated using non-blocking methods we were able to reduce the cost of halo cell communication by overlapping communication and computation of local cell neighbours. For 32 processors of the shared-memory system, halo cell communication cost dropped from 40% to under 10% of total simulation time. Again, however, in distributed-memory tests the overlap of communication and computation only improves relative speed-up from 2 to 3 on 32 processors. The success of overlapping communication and computation relies on there being sufficient ‘local’ computation to hide the communication cost. As we see in, e.g. Figure 5.8(c), the relative cost of neighbour calculation is small compared to communication and so we can only hide a small proportion of the communication behind the computation.

In spite of the limitations outlined here, we have improved the speed of simulations of our model. Figure 5.17 shows the iterative improvement in the raw speed of our algorithm for both the weak and strong scaling experiments. For a fixed-size evenly distributed simulation we obtain close to two orders of magnitude speed-up over the original serial algorithm on the shared-memory machine. Of this speed-up, around 5 times is accounted for by improving the serial efficiency of the costly neighbour calculation by using vector insertion (Section 5.3), while a further 20 times speed-up results from the optimised parallel algorithm using 32 processors of a shared-memory HPC. Consequently, for this specific problem we reduce the computation time for the user from around 2.5 hours to around 2 minutes. For simulations longer than 100 time steps this scaling would be maintained provided there are no large spatial redistributions or growth of cells. As a result this simulation is a much more tractable tool for individual-based computational modelling of cell populations. With regard to enabling the simulation of larger-scale problems, the result of the efficient use of memory for a distributed system means that we can simulate much larger problems than when using a serial algorithm.

While we carry out scaling experiments on a fixed problem that gives rise to an even distribution of load between processors, some applications of this model do not lead to such load-balance. Furthermore, the growth of populations in simulations of e.g. tumour

spheroids and cell monolayers means that the distribution of cells is unknown in advance of the simulation. Our development and implementation of a load-balance algorithm based on the localised diffusion of load between processors facilitates more efficient parallel execution of such models. For our simple growing monolayer, we reduce a measure of global imbalance from 2069.9 to 219.3 at the end in the simulation, resulting in a halving of total computation time on 8 processors. This evidence demonstrates that, with the addition of a basic load-balancing algorithm, the benefits of parallel speed-up for regularly distributed cell population simulations may be extended to a broader class of model.

Future Work

The natural extension of the work in this chapter is to further reduce the communication cost so that we can use the full power of distributed systems. For specific model applications we could achieve this by specialising the halo data transmitted between processors. Rather than serialising a complete cell, we would only send data needed to calculate interactions. This would require significant development of the Chaste infrastructure. Currently all classes that compute interactions are designed only for use with full cell objects. The use of reduced halo cell objects would require all such classes to implement parallel methods for interaction between normal cells and halo cells.

A further possible direction of investigation 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 we noted in Chapter 4, we must still copy cell data between the CPU and the GPGPU, but if such operations are faster than equivalent transfer of halo cells through the network of a distributed system this approach may enable further speed-up.

Our load-balance algorithm could be further developed in a number of ways. Firstly, 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 migration of cells between processors at each iteration, and the results in Figure 5.16 suggest that

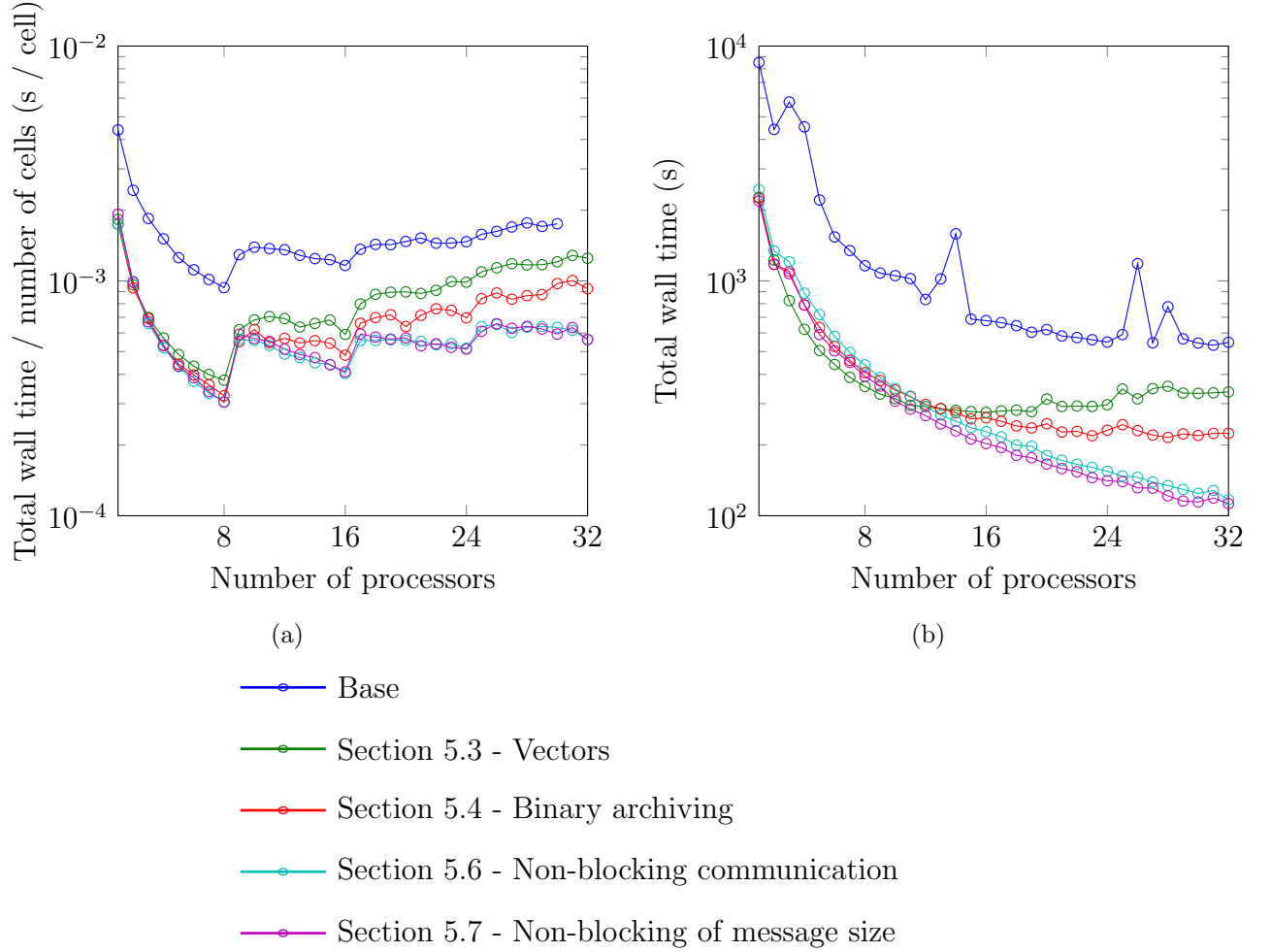


Figure 5.17: The aggregate results of optimisations described in this chapter. (a) The total simulation time divided by the number of cells for each of the iterative improvements for the weak scaling test. (b) The total simulation time for each of the iterative improvements for the strong scaling test. The base results for 31 and 32 processors of distributed-memory are unavailable due to exceeding time limits on the machine.

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. Secondly, 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 and halo cells, we may achieve an improved overall balance of computation time.

Chapter 6

Case study: the instigation of colorectal crypt fission

We demonstrate an application of our work in Chapters 4 and 5 by extending our algorithm to simulate a detailed biomechanical model of a population of cells. By introducing intra- and inter- cell mechanical heterogeneity to model the nucleus and different cell types, we investigate the role of passive cell biomechanics in the initiation of crypt fission in the colorectal crypt. Our parallel algorithm and optimisations make this model tractable by reducing computation time for a single simulation from over a week to under a day.

6.1 Introduction

The crypts of Lieberkühn are test tube shaped invaginations which descend into the lining of the small intestine, and are the location of the rapid cell proliferation that enables the epithelial layer to persistently renew itself every few days. This self renewal permits the maintenance of the epithelial layer in the toxic environment of the small intestine. While the high rate of turnover found in the intestinal epithelium has made it an interesting area of research, because it serves as a model of epithelial growth and tissue renewal, the primary reason for studying it is its association with the development of colorectal cancer

(CRC). CRC is one of the leading causes of cancer related mortality, killing in excess of 700,000 people worldwide in 2010 [Lozano et al., 2013].

Tumour development typically stems from the accumulation of rare, aberrant gene mutations that occur during DNA replication and lead to pathological cell behaviour, such as hyper-proliferation. While the increased rate of proliferation in the healthy crypt compared to other epithelial tissues would increase the occurrence of such mutations, the rapid turnover of cells (2-5 days in mice) would likely make such mutant cells short lived. Cancerous mutations that do persist are believed to spread from a single crypt to form precancerous polyps by a process of buckling in the epithelial layer and attached basement membrane, called crypt fission [Greaves et al., 2006]. Therefore, elucidating the initiating causes and biomechanical processes underlying crypt fission forms a significant part of understanding the progression of CRC, from initial mutation through to polyp formation. In this chapter we first provide an introduction to the basic biology and structure of the crypt, and the process of crypt fission. We then review the use of computational modelling in this area, before proposing an investigation into the role of passive cell biomechanics in the early stages of fission.

6.1.1 Crypt structure and fission

While crypts are dynamic environments, they maintain a broadly constant organisational structure for the epithelial layer that lines their surface. Figure 6.1 shows a cartoon and an image of a cross-section of a murine small intestinal crypt. The base of the crypt is populated by two cell types: stem cells (stained green in Figure 6.1(b)) and Paneth cells (stained red in Figure 6.1(b)). Stem cells are responsible for generating the supply of cells needed for the turnover of the epithelium. Stem cells divide producing two daughter cells that are either committed to differentiation (called transit amplifying cells) or remain stem cells in the base of the crypt, thereby maintaining their own existence [Brittan and Wright, 2004]. Paneth cells are also exclusively located at the base of the crypt, and produce and secrete anti-microbial peptides stored in granular compartments in their cytoplasm [Ganz,

2000]. While it has long been accepted that stem cells and Paneth cells reside in the base of the crypt, the precise identification of stem cells, and the spatial organisation of the two cell types is a topic of debate [Bjerknes and Cheng, 1981, Potten et al., 1997]. However, in 2007 Barker et al. [2007] identified stem cells as those expressing the *Lgr5* gene and demonstrated that stem cells are interspersed with Paneth cells at the base of the crypt, rather than residing directly above the Paneth cells as previously proposed [Potten et al., 1997].

After division from a parent stem cell, transit amplifying cells enter a phase of rapid proliferation followed by differentiation into one of four cell types: Paneth cells, which are believed to migrate to the crypt base, Goblet cells, enteroendocrine cells and enterocytes. Goblet cells and enteroendocrine cells secrete mucus and hormones respectively, while enterocytes are the most abundant cell type and are responsible for the absorption of nutrients from the intestinal lumen. With the exception of Paneth cells, the differentiated cells migrate towards the crypt orifice and to the villus tip before being sloughed into the lumen after around 2-3 days [Potten et al., 1997]. Cells that lose sufficient contact with the extra-cellular matrix undergo a form of cell death called anoikis, thus maintaining an epithelial monolayer [Frisch and Francis, 1994].

This balance of growth and migration with cell loss and death is responsible for homeostasis in the dynamic environment of the crypt and perturbations of the signaling pathways governing this balance are found in most colon cancers [Näthke, 2004]. It has been proposed that such perturbations spread to multiple crypts through crypt division events called ‘fission’ [Preston et al., 2005, Greaves et al., 2006]. During fission a single crypt divides into two or more crypts, typically by splitting at its base along its longitudinal axis, although, in adenoma, fission is also observed along the sides of the crypt [Wong et al., 2002]. Due to the difficulty in observing the fission process *in situ*, little is known about the physical processes that trigger fission, but it is not restricted to the spread of aberrant crypts: Cairnie and Millen [1975] showed that, in irradiated mice, the crypt population is re-established by the division of remaining crypts by fission, suggesting fission is also

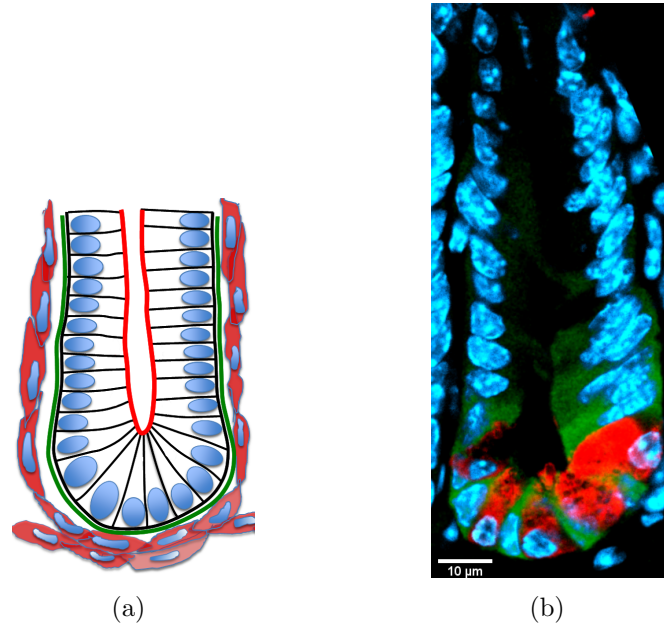


Figure 6.1: (a) The structure of the intestinal crypt: a test tube shaped invagination into the wall of the intestine, lined by a layer of columnar epithelial cells, with basally located nuclei. The crypts are surrounded by a basement membrane (green) and underlying stroma and muscle tissue. Adapted with permission from Dunn et al. [2012a]. (b) A stained image of a longitudinal section of a murine small intestine crypt, with stem cells stained green, Paneth cells stained red and cell nuclei stained blue. Image courtesy of Paul Appleton and Inke Näthke, University of Dundee.

involved in development and repair of the lining of the intestine.

The development of *in vitro* crypt structures from single stem cells by Sato et al. [2009] provides the possibility of directly observing fission events as they occur. While the geometry and forces present in the crypt do not occur in the extracellular matrix construction in which these ‘organoids’ are grown, they are seen to grow through budding and fission events similar to those seen *in situ*. This suggests that the surrounding stromal tissue and signalling cues found in the crypt are not alone responsible for the initiation of crypt buckling and fission.

6.1.2 Previous modelling work

A broad range of mathematical and computational models have been proposed, each tailored to the specific hypothesis under question, to study the shape changes and buckling

behaviour exhibited by layers of epithelial cells and specifically the epithelial layer of the colorectal crypt. Here we review examples from the modelling literature of mathematical and computational modelling of epithelial layer buckling and instability, then turn our focus specifically to models of the crypt.

6.1.2.1 Buckling and instability of epithelial layers

Individual-based modelling of epithelial folding in the context of development predates computational approaches. Lewis [1947], with the aid of Miller, constructed a physical model of a row of adherent columnar epithelial cells. The cells comprised two brass bars, connected by a single horizontal bar through their midpoint attached by a peg to allow pivoting. The two ends of the bars were connected by rubber bands, at the apical and basal surfaces. The model was used to show that folding patterns of epithelial sheets seen in development can be explained by differential tension at the apical and basal surfaces of the cells produced in the model by different tensions in the rubber bands.

Odell et al. [1981] built on these ideas using a computational model to show that morphologies arising in the folding of cell layers in early development can be explained by a simple active process within each cell that causes a difference in apical and basal tension. The authors model an epithelial cell in apical-basal cross-section as a quadrilateral, with the four vertices connected by viscoelastic springs. Each vertex is shared with a neighbouring cell, and the spring spanning the apical surface is supposed to respond to extension beyond a set threshold by a strong contraction, modelling active contraction in the apical filament bundle. By setting the apical region of a single cell to be contracted as an initial condition, and considering different initial cell arrangements, the authors demonstrate qualitative agreement with observed morphogenesis in gastrulation in Echinoderm, central furrow formation in *Drosophila* and neurulation in amphibians.

More recent models have considered the folding of a 2D epithelial sheet into 3D structures during development. For example, Osterfield et al. [2013] used a modified vertex dynamics model to study tissue shape changes during appendage formation in *Drosophila*.

As described in Chapter 2, the vertex dynamics model represents the apical surface of a tightly packed cell layer, with each cell defined by the vertices of its apical region. Although initially developed as a 2D model, the authors allow vertices of a 2D layer to move in 3D perpendicular to the sheet. Different cell types are defined by scaling relative parameters that govern the resistance of vertex cells to deviations from preferred values for their area and perimeter. Furthermore, the authors introduce differing line tensions between vertices of cells in a pattern representing the unfolded appendage primordium. As a result of these differential tensions between different cell types the flat 2D configuration becomes unstable to small perturbations perpendicular to the plane and in simulations an initially flat layer is observed to buckle in 3D, and, for the patterns of cell types proposed, 3D structures analogous to the *Drosophila* appendages are formed. This individual-based model demonstrates that folding patterns in developmental epithelial layers can be explained by differences in the adhesive and mechanical properties of cells within an epithelial sheet as opposed to active processes, for example.

Analytic biomechanical models have also been widely used in studies of development. For example, Ben Amar and Jia [2013] recently published a model of the formation of herringbone patterns of protrusions on the luminal surface of the avian duodenum called villi. The proposed mechanism is the differential growth of different layers in the duodenum. The walls of the duodenum are modelled as three layers atop one another: a stiff outer epithelial layer (assumed to be planar for simplicity), a central growing elastic mesenchyme layer, and a mucosal epithelium that faces the lumen. As a result of differential growth parallel to the layer between the mesenchyme and luminal epithelium, buckling instabilities that form herringbone patterns akin to those observed *in situ* are found. A natural next step in this work would be to extend the model to the formation of crypts as well as villi.

In a similar stratified mathematical model, Hannezo et al. [2011] consider the balance of stresses that result from cell division and death in the physiological (rather than developing) intestine. They model the underlying stroma as a thick elastic medium below

a thin ($\approx 1\mu\text{m}$ thick) elastic basement membrane with a bending modulus that stabilises the cell monolayer ($\approx 10\mu\text{m}$ thick) that is directly above it. In 1D, simple wave-like buckling of the epithelial layer results from the pressure of cell division. In 2D however, a number of possible morphologies arise that mimic those seen in different regions of the intestine. When there is a large dependence of the cell layer growth rate on the local curvature of the layer, the surface obtained is similar to that in the colon, where no villi exist, whereas for curvature independent growth, small intestine patterns of crypts and villi are seen. The herringbone pattern of villi seen by Ben Amar and Jia [2013] is also recovered in this model. These two continuum models demonstrate that different growth patterns at the tissue scale can also account for the development of buckling and folding patterns in developmental epithelial layers.

In the context of tumour invasion, Basan et al. [2011] develop a stratified model of instabilities of a growing epithelial tissue that does not include a representation of the basement membrane. Based on the observation that, over long time scales, tissues behave as viscous liquids, they model the epithelial layer as an incompressible fluid that has a surface friction interaction with an underlying viscoelastic stromal layer. By considering small perturbations of the epithelial-stromal interface in the presence of the growth of the epithelial layer, their model predicts the development of finger-like protrusions of the epithelium into the stroma. Specifically, they find that the interface is destabilised by an increase in cell growth rate or the size of the growing region (which may be caused by cancerous genetic mutations) or by a reduction of the interfacial tension between the epithelium and stroma, possibly as a result of degradation of the basement membrane due to the secretion of proteases by cancerous cells.

6.1.2.2 Buckling and instability in the crypt

Biomechanical models specific to the crypt can be split into two broad categories: continuum models based around buckling of a beam, and individual-based cell centre models.

Edwards and Chapman [2007] developed a continuum model of budding and fission events in the colorectal crypt based on the beam equation. The epithelial layer is modelled by a thin beam, which has a central growing region. The beam is assumed to be attached to an underlying basal lamina by viscoelastic springs which act to stabilise the layer, but allow the migration of cells parallel to the layer. By considering the effect of altering parameters for the growth and motility of cells, the authors find that the flat steady state can be destabilised, and hence assumed to buckle, by an increase in either the growth rate of cells or the size of the proliferative region, or by changes in adhesion and motility in cells that acts to limit their migration. Biologically, this model suggests that changes in the regulation of cell proliferation and the adhesion of cells to each other and the basement membrane are the principal factors that initiate buckling of a normally smooth lining of a crypt.

Nelson et al. [2011] extend the model of Edwards and Chapman [2007] to aid the interpretation of *in vitro* experiments. They model the deformation out of the plane observed in a monolayer of epithelial cells growing on a substrate. They use a bilayer model, where each cell is represented as a spring and the substrate as a thin elastic beam, to uncover the buckling modes in the experimental system. Furthermore the model and analysis of Edwards and Chapman is extended to consider the longer time scale relaxation of the spring stresses between the epithelium and stroma that result from buckling. In this analysis the initial buckling instabilities uncovered by Edwards and Chapman are shown to develop into crypt-like geometries which suggests that the process that leads to buckling uncovered by Edwards and Chapman may also be implicated in the developmental process that generates crypts.

Drasdo [2000] developed an individual-based cell centre model of single layered cell populations that shows Eulerian buckling can result from an imposed bending energy of the cell layer being overcome by high cell proliferation rates and limited migration due to strong friction. Building on earlier work by Drasdo et al. [1995], the author models single cells as spheres, or dumb-bells directly after mitosis, and proposes energetic costs

associated with cell neighbour separation and overlap, bending of the cell layer and cell mitosis along axes not parallel to the local tangent of the cell layer. A Metropolis-Hastings algorithm [Metropolis et al., 1953] is used to evolve the state of the population, where cells migrate, grow or change their orientation to minimise total energy in the system. The instabilities that arise are maintained only so long as cell proliferation is active, and dissipate if proliferation is stopped. This mechanism is specifically proposed to apply in intestinal crypts when hyper-proliferation that cannot be overcome by cell migration and sloughing into the lumen leads to aberrant buckling of the epithelial layer in the crypt.

By replacing the Metropolis algorithm of Drasdo [2000] in favour of a Langevin system of equations for the dynamics of cell motion and orientation in a force-based approach, Drasdo and Loeffler [2001] produce similar conclusions to Edwards and Chapman [2007] using an individual-based off-lattice model with a crypt-like test tube geometry imposed underneath the cell layer. They show that buckling of a cross-sectional 2D model of a crypt is the result of cell division beyond the rate that may be balanced by cell migration towards the crypt orifice overcoming the bending rigidity of the layer. These individual-based models concur with the conclusions of Edwards and Chapman and Nelson et al., that the buckling of the epithelial layer in early colorectal crypt fission may occur as a result of the breakdown in the balance of cell proliferation and cell migration along the crypt-villus axis.

A growing body of work in the development of agent-based models of epithelial layers is focused on integrating the role of the basement membrane and extra-cellular matrix. While the basement membrane has many roles beyond simply anchoring cells, this work has primarily focused on the biomechanical nature of the interaction between epithelia and the membrane.

Buske et al. [2012] recently extended their earlier cell centre model of spatio-temporal organisation in the crypt [Buske et al., 2011] to study the development of crypt organoid cultures developed by Sato et al. [2009]. While their earlier model defines a crypt-like geometry by a fixed triangular network that represents the basement membrane, to model

organoid growth in a gel substrate they consider the triangular network as a dynamic, flexible, closed surface. The membrane can grow as a result of tension produced by proliferating epithelial cells attached to its inner surface. When simulated, the model is shown to produce buckling behaviour as a result of instability caused by cell proliferation, in a similar manner to the 2D model of Drasdo and Loeffler [2001], and the formation of stable crypt-like structures observed experimentally are seen in the model [Sato et al., 2009]. Further, by assuming a dependence of Paneth cell specification from proliferative cells on the local curvature of the membrane, patterns of Paneth and proliferative cells in agreement with experimental observation are found. This model suggests that the same mechanism uncovered by Drasdo and Loeffler and the work of Edwards and Chapman [2007] and Nelson et al. [2011] for a crypt also applies to the 3D organoid cultures.

By adapting the off-lattice crypt organisation models of Meineke et al. [2001] and van Leeuwen et al. [2009], Dunn et al. [2012b] develop a model of a dynamic growing epithelial layer that includes an explicit representation of the basement membrane and underlying stromal cells. Rather than the luminal surface of an unrolled cylinder considered in the earlier papers, the authors consider a cross-section through an epithelial tissue layer and model individual epithelial and stromal cell centres as points in space. Cell connectivity and geometry is defined by the Delaunay triangulation of the centres and its dual, the Voronoi diagram, respectively. The basement membrane is modelled implicitly as a piecewise linear curve through the midpoints of springs connecting adjacent epithelial and stromal cells. Each epithelial cell is assigned a random time until its next division, during which its mitotic axis is aligned parallel to the local portion of the basement membrane. An additional force is also introduced which acts to maintain zero curvature of the basement membrane by applying a force to epithelial cells in regions where the local curvature of the basement membrane is non-zero. Dunn et al. [2012a] extend this model to develop a deformable model of the crypt epithelial layer, basement membrane and surrounding stroma by instead assigning a non-zero target curvature. The result of this spontaneous curvature is that an initially flat epithelial layer deforms into a test tube shaped invagina-

tion into the underlying stroma that resembles the structure of the crypt. This approach has the potential to incorporate the role of the surrounding tissue during crypt fission, which is of interest as it is not only the epithelial layer that is deformed by fission, but also the surrounding tissue.

In the context of modelling the basement membrane, D’Antonio et al. [2013] recently published a development of the model of ductal carcinoma *in situ* by Macklin et al. [2012] that extends a cell centre model to include a discrete representation of the basement membrane. In this case the membrane is modelled explicitly as a collection of points that interact with each other, the underlying extra-cellular matrix and individual epithelial cells through a spring potential. The model is further extended to incorporate a representation of long time scale plasticity seen in biological tissue but overlooked by simple visco-elastic models. By allowing sustained deformation to alter the equilibrium length of springs between basement membrane and extra-cellular matrix, the natural state of a tissue may be altered over time. The model is shown to produce results consistent with observed morphology of epithelial cell layers, and future work will look to apply the model to understanding the mechanisms underlying tumour cell breach and invasion through the basement membrane.

In addition to the mechanically motivated models considered here, a further body of work has developed using off-lattice models of the crypt to focus on the organizational structure of homeostatic crypts and the process of monoclonal conversion, in which the progeny of one stem cell repopulates the entire crypt and all other stem cell clones are lost. This process is repeated periodically, leading to continual turnover of the stem cell population. This is of relevance to CRC since monoclonal conversion facilitates the propagation of mutant cell populations within neoplastic crypts, prior to crypt buckling and fission [Meineke et al., 2001, van Leeuwen et al., 2006, Fletcher et al., 2012, Mirams et al., 2012].

In summary, mathematical and computational models have been developed to model the shape changes and buckling of epithelial cell sheets in both developmental and patho-

logical settings. Broadly speaking, these models suggest that buckling instabilities result from cell growth and proliferation which cannot be balanced by cell migration. Furthermore, specific folding and buckling patterns observed in developmental processes can result from differences in the individual adhesive and mechanical properties of cells.

6.1.3 Aim of work

Arising from discussions and collaboration with the Näthke lab at the University of Dundee, the aim of this chapter is to investigate the effect of subcellular biomechanical heterogeneity on the initiation of crypt fission. Previous models of crypt fission ignore the columnar shape of epithelial cells, and the subcellular mechanical properties of their different compartments. The epithelial layer is either modelled as a thin beam, in the case of Edwards and Chapman [2007] and related work, or as a piecewise linear connection of cell centres where all cell-cell forces are transmitted through the centre of mass of the cell, as in Dunn et al. [2012a].

Both assumptions ignore the columnar shape of cells in epithelial layers. Indeed, aspect ratios of between 2 and 4 have been observed in cross-sectional samples of murine crypts (Näthke and Appleton, private communication and visible in Figures 6.2(a) and 6.2(b)). In cell centre models the stability provided by the columnar shape is instead introduced by a hypothetical bending energy. Recent individual-based approaches to studying fission are based on the models of Meineke et al. [2001] and Buske et al. [2011], which were proposed to model the dynamic organisation of the large intestinal crypt. In these models a detailed representation of the physics of cell-cell interactions is not important, and simple adhesion and volume exclusion is sufficient and a significant improvement over lattice-based approaches. Furthermore, Paneth cells are not found in the large intestine studied in these models, suggesting a more homogeneous population in the crypt lining. As noted above, it is questionable, however, whether such models are applicable to the detailed study of biomechanical processes, and in the small intestine where Paneth cells are found. Furthermore, continuum models fail to account for any of the subcellular or

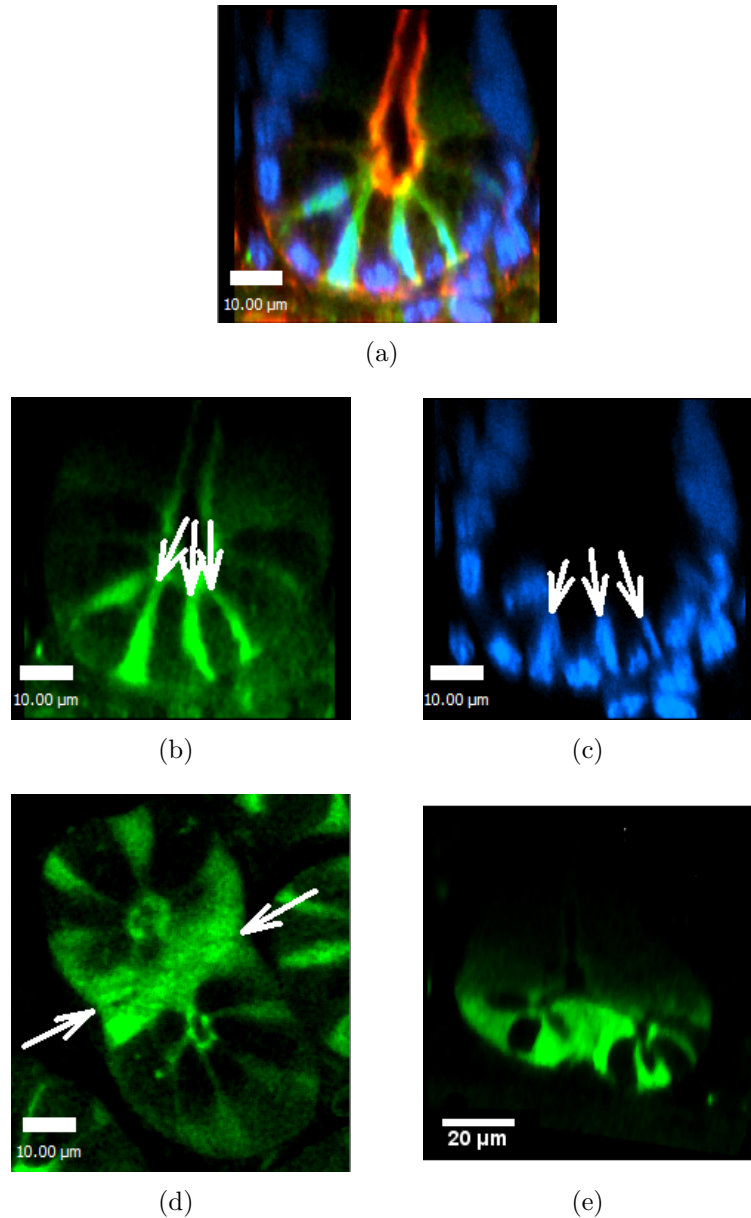


Figure 6.2: Images of murine crypts obtained by Appleton and co-workers. Murine crypts were fixed, stained and imaged using a confocal microscope. (a) A longitudinal section of the base of a homeostatic murine small intestinal crypt. Stem cells are stained green, nuclei blue and F-actin at the luminal surface (red / orange). (b) A longitudinal section of the base of a homeostatic murine small intestinal crypt. Stem cells are stained green and all cells can be seen to be highly columnar. (c) A longitudinal section of the base of a homeostatic murine small intestinal crypt. Cell nuclei are stained blue, and stem cell nuclei (white arrows) can be seen to be highly compressed. (d) A transverse section through a fissioning murine small intestinal crypt. Stem cells are stained green, and can be seen to be clustered along the buckling point. (e) A longitudinal section through a fissioning murine small intestinal crypt. Stem cells are stained green, and can be seen to be clustered along the buckling point. Images courtesy of Paul Appleton and Inke Näthke, University of Dundee.

intercellular variety in mechanical properties except for in an averaged sense. A more realistic biomechanical model of cells is given by the ScEM, which has been shown to produce more than just a caricature of cell biomechanics and further allows to model subcellular, as well as intercellular, mechanical heterogeneity [Sandersius and Newman, 2008].

Using this model, we first show that a layer of columnar ScEM cells are stable under lateral compression, in contrast to a layer of cells modelled simply by cell centres. This poses a further question: what causes the initial perturbation that may destabilise an otherwise stable layer of columnar epithelial cells and initiate buckling?

Näthke and co-workers have observed that buckling events in *in vitro* crypt organoid cultures typically occurs as stem cells become compressed down to the width of their nuclei as a result of mitotic pressure. This level of compression can also be seen in *in vivo* imaging, such as Figure 6.2(c), where the stem cell nuclei (cyan) are more highly compressed than those of neighbouring Paneth cells. While no data are yet available for intestinal cells, it has been shown that the nucleus of other eukaryotic cells is significantly stiffer than the surrounding cytoplasm. By compressing bovine endothelial cells between two glass microplates, Caille et al. [2002] estimated that the elastic modulus of the nucleus was approximately ten times that of the cytoplasm. In micropipette aspiration experiments, Maniotis et al. [1997] found bovine endothelial nuclei to be ten times stiffer, Dong et al. [1991] also found a ten-fold increase in stiffness in neutrophils and Guilak et al. [2000] found an approximate four-fold difference in pig chondrocytes. In light of these data and the observations in organoid cultures, we propose that the increased stiffness of cell nuclei in a cell layer under compression could provide an initial perturbation that may trigger buckling.

It has also been observed by Näthke and Appleton that fissioning events in *in vitro* crypt organoid cultures tend to occur when two or more stem cells are positioned adjacent to one another, rather than being evenly dispersed between Paneth cells. This can be seen in Figure 6.2(d), where stem cells (labelled green) are visibly clustered around a

buckling point in a transverse section of a buckling murine small intestinal crypt, and in a longitudinal section in Figure 6.2(e). We use our model to investigate whether differences in the sizes and inter- and intra- cellular mechanical properties of Paneth and stem cells could explain this observation. Although measurement of cross-sectional cell dimensions indicates that Paneth cells typically have double the area of stem cells, in the absence of rheology data for these cell types we may only hypothesise about their relative bulk properties.

In the remainder of this chapter, we carry out computational experiments to investigate these hypotheses. We model the lateral compression of a layer of cells to reproduce the increasing pressure as a result of cells growing and dividing in the cell layer. Although we do not explicitly model cell growth and division, by incrementally increasing the level of compression we are able to represent the change from homeostatic conditions where cell migration balances proliferation, to a case where excess proliferation causes a substantial compression of cells.

6.2 Materials and methods

This section sets out our approach to simulating the initiation of buckling. In Section 6.2.1 we present the details of the mechanical ScEM developed by Newman and co-workers [Newman, 2005, Sandersius and Newman, 2008]. This model allows the inclusion of a more detailed biomechanical representation of the cell and its subcellular components than is possible with a simple cell centre model. In Section 6.2.2 we describe our approach to extending the ScEM to incorporate different cell types, by assigning relative areas and relative bulk stiffnesses to cells. We also describe our approach to modelling the cell nucleus as a distinct subcellular region that is stiffer than the surrounding cytoplasm. Finally, in Section 6.2.3 we discuss pertinent details of implementing the ScEM in Chaste to make use of the parallel algorithm developed in Chapters 4 and 5 which makes simulating this more detailed model computationally tractable.

6.2.1 The subcellular element model

As we noted in the introduction, the ScEM was primarily developed so that, rather than being imposed within a model, a cell's shape is an emergent property of its interaction with the local environment. This is achieved by partitioning a model cell into many small volumes of cytoplasm (or other organelles) called elements. Each element (which we denote using Greek indices) α_i of a cell (which we denote using Roman indices), i , is modelled as a single point at its centre of mass, $\mathbf{x}_{\alpha_i}$, which changes over time subject to three processes: (i) weak random fluctuations, (ii) elastic interaction with elements β_i of the same cell and (iii) elastic interaction with elements β_j of other cells. The motion of each element is assumed to be subject to strong viscous drag due to surrounding cytoplasm, which is characterised by a drag coefficient η_i . It is assumed that viscous terms dominate inertial terms, so that we may ignore inertial terms in the equation of motion for each element. Elastic interactions between elements are characterised by phenomenological potentials V_{intra} for elements of the same cell and V_{inter} for two elements of different cells. The resulting equation of motion for each element α_i is given by

$$\eta_i \dot{\mathbf{x}}_{\alpha_i} = \boldsymbol{\xi}_{\alpha_i} - \nabla_{\alpha_i} \sum_{\beta_i \neq \alpha_i} V_{\text{intra}}(|\mathbf{x}_{\alpha_i} - \mathbf{x}_{\beta_i}|) - \nabla_{\alpha_i} \sum_{j \neq i} \sum_{\beta_j} V_{\text{inter}}(|\mathbf{x}_{\alpha_i} - \mathbf{x}_{\beta_j}|), \quad (6.1)$$

[Sandersius and Newman, 2008] where $\boldsymbol{\xi}_{\alpha_i}$ is a Gaussian noise term drawn from the same distribution for each element.

Within this formulation, the biomechanical properties of the model cells are encoded in the potential functions V_{intra} and V_{inter} . Sandersius and Newman [2008] choose a form of the potential akin to the Morse potential [Morse, 1929], originally developed to describe diatomic molecules, as a convenient formulation of the close-range repulsion and medium-range attraction of elements of cytoskeleton. Sandersius and Newman [2008] suggest that the precise functional form of the potential has little impact on the system dynamics for the ScEM, which is in agreement with the findings of Pathmanathan et al. [2009] for bulk properties of similar models at the tissue scale. The Morse potential for two subcellular

elements whose centres lie a distance r apart is

$$V(r) = u_0 \left(e^{2\rho \left(1 - \frac{r^2}{r_{\text{eq}}^2}\right)} - 2e^{\rho \left(1 - \frac{r^2}{r_{\text{eq}}^2}\right)} \right), \quad (6.2)$$

where u_0 and ρ are tunable parameters and r_{eq} is the equilibrium distance between elements. This gives rise to a force $F(r)\hat{\mathbf{r}}_{\alpha_i\beta_j}$ between pairs of elements α_i and β_j , where $\hat{\mathbf{r}}_{\alpha_i\beta_j}$ is the unit vector joining their centres and

$$F(r) = -\frac{\partial V}{\partial r} = \frac{4\rho u_0 r}{r_{\text{eq}}^2} \left(e^{2\rho \left(1 - \frac{r^2}{r_{\text{eq}}^2}\right)} - e^{\rho \left(1 - \frac{r^2}{r_{\text{eq}}^2}\right)} \right). \quad (6.3)$$

For small deviations from the equilibrium distance r_{eq} , the force may be approximated by that of a linear spring with spring constant

$$k_0 = \frac{8\rho^2 u_0}{r_{\text{eq}}^2}, \quad (6.4)$$

as shown in Appendix A. We specify the value of the spring constant k_0 here, and derive the value u_0 for each cell at the beginning of a simulation using equation (6.4). The force magnitude $F(r)$ is plotted as a function of r/r_{eq} in Figure 6.3 for two relative values of u_0 .

To use the ScEM to model cell biomechanics we require that the properties of the cell are independent of the number of elements used to model it. This is achieved by scaling the parameters r_{eq} , u_0 and η_i . A full derivation of the scaling for the spring constant is presented in Appendix A, while the scalings are stated here for reference. For a cell composed of N subcellular elements, that forms a sphere (circle in 2D) of radius R_{cell} , the equilibrium distance is chosen to be

$$r_{\text{eq}} = \begin{cases} 2R_{\text{cell}} \left(\frac{p_3}{N} \right)^{\frac{1}{3}}, & \text{in 3D,} \\ 2R_{\text{cell}} \left(\frac{p_2}{N} \right)^{\frac{1}{2}}, & \text{in 2D,} \end{cases} \quad (6.5)$$

where $p_3 = \pi/3\sqrt{2}$ and $p_2 = \pi/2\sqrt{3}$ are packing densities for spheres and circles respec-

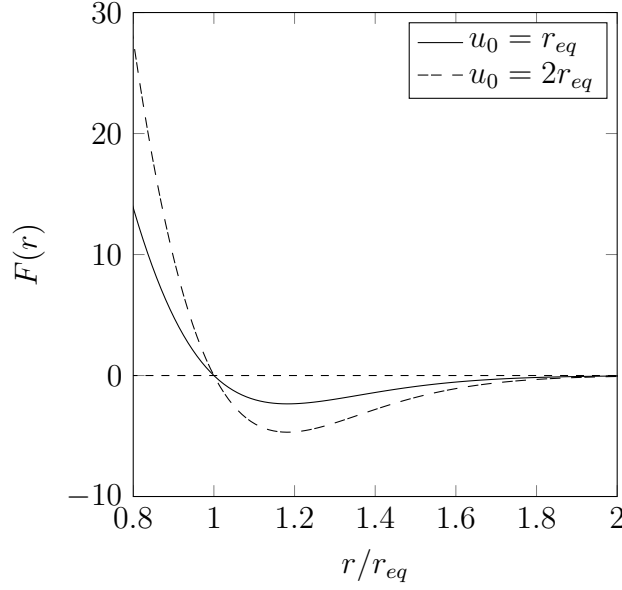


Figure 6.3: The scaled interaction force between elements separated by a distance r for a given equilibrium distance r_{eq} . Plots show two different relative values of the parameter u_0 , which is proportional to the bulk stiffness of the cell. Parameters used: $N = 1024$, $R_{cell} = 5\mu\text{m}$, $\rho = 2.0$. Positive values of $F(r)$ represent repulsion, while negative values represent attraction.

tively [Sandersius and Newman, 2008]. The spring constant k_0 , and hence the depth of the potential u_0 , is scaled by $N^{-\frac{1}{3}}(1 - \lambda N^{-\frac{1}{3}})$ in 3D and $(1 - \lambda N^{-\frac{1}{2}})$ in 2D. The parameter λ is introduced by Sandersius and Newman [2008] to tune this scaling factor and, by simulating creep strain experiments on cells composed of different numbers of elements, was found to have an optimal value of approximately 0.5. Finally, the viscosity η_i is scaled by $1/N$.

6.2.2 Model development

We extend the ScEM by defining a subregion of each cell to be the nucleus in order to model its increased bulk stiffness. Furthermore, we define cell types which may differ in both their bulk mechanical properties and their area to model the Paneth and stem cells found in the base of the crypt. Since all simulations for this application are 2D, we describe our model development for this case, but the approaches and code produced are extensible to 3D with little alteration.

6.2.2.1 Cell nucleus

We identify the cell nucleus by labeling a subset of a cell's elements as nuclear elements. By scaling the potential for interactions between two nuclear elements of the same cell by a constant $K_{\text{nuc}} > 1$ we obtain a stiffer subregion of each cell. The value of K_{nuc} is fixed for all cells and independent of the values K_i so that the resulting potential between two nuclear elements of the same cell is given by

$$V_{\text{intra nuc}}(r) = K_{\text{nuc}} u_0 \left(e^{2\rho \left(1 - \frac{r^2}{r_{\text{eq}}^2}\right)} - 2e^{\rho \left(1 - \frac{r^2}{r_{\text{eq}}^2}\right)} \right). \quad (6.6)$$

Although our model does not define the nucleus to be a connected subregion of the cell, we find that for values in line with observed experiment ($K_{\text{nuc}} \approx 10$ [Maniotis et al., 1997]) the increased attractive force between nuclear elements and the initial condition enforces this condition.

6.2.2.2 Biomechanical cell types

We define different cell types in the ScEM based upon their relative stiffness and their area. Noting that the elastic modulus of a model cell is proportional to the parameter u_0 in the interaction potential in equation (6.2), we define a cell i 's relative stiffness by a positive constant K_i . The same value of u_0 is fixed for all cells and we modify the interaction potential between elements of the same cell, i , to be

$$V_{\text{intra}}(r) = K_i u_0 \left(e^{2\rho \left(1 - \frac{r^2}{r_{\text{eq}}^2}\right)} - 2e^{\rho \left(1 - \frac{r^2}{r_{\text{eq}}^2}\right)} \right). \quad (6.7)$$

Figure 6.3 shows how, for values $K_i > 1$, the magnitude of the interaction force is increased by a factor K_i for any given separation. We assume that attraction will be weaker between elements of different cells than between elements of the same cell and therefore choose a relative scaling constant $K_{\text{diff}} = 0.5$ for interactions between elements of distinct cells, so that

$$V_{\text{inter}}(r) = K_{\text{diff}} u_0 \left(e^{2\rho\left(1-\frac{r^2}{r_{\text{eq}}^2}\right)} - 2e^{\rho\left(1-\frac{r^2}{r_{\text{eq}}^2}\right)} \right). \quad (6.8)$$

6.2.2.3 Differential cell area

Cell area A is related to the model parameters N and r_{eq} by the equation

$$A = \frac{N\pi r_{\text{eq}}^2}{4p_2}$$

and hence, to vary cell area, we may either change the number of elements in a cell or change the equilibrium distance. For reasons of computational simplicity we proportionally reduce the number of elements in the cell and rescale η_i and u_0 appropriately so that the bulk properties of the cell remain unchanged. For interactions between two elements of cells of different sizes we use the mean value of the scaled parameter u_0 for each cell in equation (6.8).

6.2.2.4 Boundary conditions and compression

We approximate surrounding tissue, such as the basement membrane and stroma by hard-wall boundary conditions. We use a fixed boundary condition along the x -axis: any element located at $\mathbf{x}_{\alpha_i} = (x_{\alpha_i}, y_{\alpha_i})$ for which $y_{\alpha_i} < 0$ is projected to $(x_{\alpha_i}, 0)$.

To represent growth and proliferation of surrounding cells we compress the cell layer by fixing two hard-wall boundary lines parallel to the y -axis at the extremities of the layer and we incrementally move the lines together at a rate C so that a cell layer of initial width W has width $W(1 - Ct)$ at time t . As we note in Appendix A.3, the viscous timescale for model cells is approximately 1 s, therefore the compression is carried out at 2 s intervals to allow the configuration to relax. At the compression intervals we re-scale all x -co-ordinates of cell elements by $1 - 2C$ so that stress is evenly distributed through the layer, rather than focussed at the layer extremities where it will cause large deformation of the boundary cells.

6.2.3 Implementing the ScEM in Chaste

To simulate the ScEM we extend the implementation in Chaste for simulating the overlapping spheres model described in Chapter 3. This is an example of the benefit of Chaste's design as a modular library: much of the functionality including setup, output and the parallel algorithm, as well as the necessary data structures are already implemented and well tested. Consequently, implementing the ScEM is reduced to defining a new force class **SemForce**, children of the mesh and population classes for overlapping spheres (**SemMesh** and **NodeBasedCellPopulationWithSemId**) and two helper classes: a class (**SemParameterScaler**) to scale parameters η_i , u_0 and r_{eq} , and a class (**SemCellsGenerator**) to enable the construction of a population of ScEM cells from a single archive of a cell's initial element locations. The archives for ScEM cells are created with a standalone Python script using an algorithm to place elements close to equilibrium. All classes in Chaste are templated over spatial dimensions, making the transition between 2D and 3D models straightforward.

Creating ScEM cells

To generate a 2D ScEM cell consisting of N elements and of area πR_{cell}^2 inside a region Ω (also of area πR_{cell}^2) we proceed as follows. A location $\mathbf{r}_{\alpha_0} \in \Omega$ is chosen for element α_0 . A possible location $\tilde{\mathbf{r}}_{\alpha_1}$ for element α_1 is selected uniformly at random in Ω and accepted if $\|\tilde{\mathbf{r}}_{\alpha_1} - \mathbf{r}_{\alpha_0}\| \geq 0.75r_{eq}$. This process is iterated, accepting $\tilde{\mathbf{r}}_{\alpha_k}$ only if $\|\tilde{\mathbf{r}}_{\alpha_k} - \mathbf{r}_{\alpha_j}\| \geq 0.75r_{eq}$ for all $j < k$, until all elements are assigned a location. This is implemented in a standalone Python [Python, 2013] script.

Force class

The **SemForce** class has member variables **mSpringStiffness**, which corresponds to k_0 in equation (6.4), **mNuclearSpringFactor** corresponding to K_{nuc} in equation (6.6) and a map from cell indices i to relative cell stiffnesses K_i in equation (6.7). As a member of the mechanics class hierarchy in Chaste, **SemForce** has a method **AddForceContribution()**,

which takes a reference to the cell population as an input argument. From the cell population it obtains a list of neighbouring elements and the force between pairs of elements is evaluated using the appropriate equation from equations (6.6)-(6.7), depending on the element types.

Parameter scaling

Since scaling of the parameters depends on the number of elements per cell, and is used in multiple classes throughout the simulation, we implement a helper method using the singleton design pattern [Gamma et al., 1994]. This ensures that all parameters are scaled consistently across all classes, and brings all scaling into a single code source file, reducing the likelihood of introducing bugs. The class contains a map from cell indices to the number of elements contained in the cell, as well as members for the constants ρ and λ . Two methods accept a cell index and an unscaled damping constant η_i or unscaled spring constant k_0 and return the scaled value of η_i and u_0 respectively. A further method returns the equilibrium distance r_{eq} .

Mesh and cell population classes

The mesh class `SemMesh` inherits from the `NodesOnlyMesh` and differs only by correctly setting the cell and nucleus labels of elements. The cell population child class overwrites output methods to write ScEM cell indices to file in addition to location data. The method for updating element locations is modified to take into account the scaled damping constant η_i in the integration scheme.

Generating cell populations

The class `SemCellsGenerator` constructs initial conditions by accepting the location of a cell archive file generated by the Python scripts and a translation co-ordinate for each cell. A copy of each cell from the archive is then created with its centre of mass translated appropriately. This class also defines the initial nuclear region of a cell as a sphere (or

circle) or radius R_{nuc} , centered at $\mathbf{r}_{\text{nuc}}$ in the untranslated cell.

Gaussian noise

We implement the Gaussian noise in equation (6.1) in the force class `SemDiffusionForce` as a random perturbation to the force on each element at each time step. The perturbation at time t to the force on an element α_i is given by $\boldsymbol{\xi}_{\alpha_i}(t) = (\xi_{\alpha_i}^m(t))$ where for all $m = 1, 2$, $\xi_{\alpha_i}^m(t) \sim N(0, \eta_i \sqrt{2D/\Delta t})$, where η_i is the viscosity, Δt the integration time step and D is a diffusion parameter.

6.3 Results

We carry out three compression experiments. First, we consider the dynamics of cell nuclei in a layer of model cells under compression for different values of the nuclear stiffness K_{nuc} . Second, we consider the effect of the different size of Paneth and stem cells observed in the crypt on the buckling of a layer of cells. Finally, we consider whether altering the relative stiffness of Paneth and stem cell cytoplasm significantly alters these results. The common parameter values for each experiment are given in Table 6.1.

In each simulation we consider two metrics to characterise the level of buckling seen within the layer at time t . First, for the buckled state of the whole cell layer, we consider the standard deviation of the height above the basal boundary condition of the centre of mass of each cell, given by

$$\sigma_{\text{cell}} = \sqrt{\frac{\sum_{i=0}^{N_{\text{cell}}-1} (\bar{y}_i - \bar{y})^2}{N_{\text{cell}}}},$$

where $\bar{y}_i$ is the mean height of elements in cell i above the basal boundary condition at time t and $\bar{y}$ is the mean height of all elements above the basal boundary condition at time t . Second, we consider the standard deviation of the height above the basal boundary condition of the centre of mass of each cell nuclei, given by

Parameter	Description	Value	Reference
r_c	Force cut-off distance	$1 \mu\text{m} (\approx 2r_{\text{eq}})$	–
η_i	Element damping constant	$10^{-9} \text{ N s } \mu\text{m}^{-1}$	Sandersius and Newman [2008]
k_0	Effective spring constant	$10^{-9} \text{ N } \mu\text{m}^{-1}$	–
ρ	Elastic potential constant	2.0	Sandersius and Newman [2008]
λ	Elastic scaling parameter	0.5	Sandersius and Newman [2008]
N	Number of elements per cell	1024	–
D	Diffusion constant	$10^{-1} \mu\text{m}^2 \text{ s}^{-1}$	Sandersius and Newman [2008]
R_{cell}	Radius of isolated cell	$5 \mu\text{m}$	–
p_2	Packing density of circles	$\pi/2\sqrt{3}$	Sandersius et al. [2011]
Δt	Integration time step	10^{-5} s	–

Table 6.1: Parameter values for the cell layer experiments. Values of N and R_{cell} are given for larger cells. For smaller cells R_{cell} is halved and consequently N is approximately halved. Parameters k_0 and η_i are the unscaled values.

$$\sigma_{\text{nuc}} = \sqrt{\frac{\sum_{i=0}^{N_{\text{cell}}-1} (\bar{y}_{\text{nuc},i} - \bar{y}_{\text{nuc}})^2}{N_{\text{cell}}}},$$

where $\bar{y}_{\text{nuc},i}$ is the mean height of nuclear elements in cell i above the basal boundary condition at time t and $\bar{y}_{\text{nuc}}$ is the mean height of all nuclear elements above the basal boundary condition at time t .

6.3.1 Effect of the cell nucleus

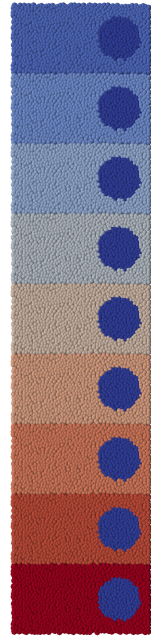
We first consider the effect of nuclear stiffness in the absence of any other variation by compressing a uniform layer of cells, and varying the parameter K_{nuc} governing the relative stiffness of the nucleus. Based on the observations of Näthke and Appleton that, in non-fissioning crypts, Paneth cells have an aspect ratio of approximately 2:1 (visible in Figure 6.2(b)), we equilibrate 9 adjacent cells with an area of $25\pi\mu\text{m}^2$, composed of 1024 elements such that they are each $6.26\mu\text{m}$ in width and $12.53\mu\text{m}$ in height. On the further observation that Paneth cell nuclear diameter is approximately two thirds of the cell width in cross-section and located basally, we set $R_{\text{nuc}} = 2\mu\text{m}$ and $\mathbf{r}_{\text{nuc}} = (0\mu\text{m}, -3.5\mu\text{m})$. The initial condition is visualised in Figure 6.4(a). We consider three values for the relative nuclear stiffness K_{nuc} : the control case $K_{\text{nuc}} = 1.0$, where the nuclear region

is biomechanically indistinguishable from the cytoplasm; the lower value found in the literature, $K_{\text{nuc}} = 5.0$; and the upper value $K_{\text{nuc}} = 10.0$ [Caille et al., 2002, Maniotis et al., 1997, Dong et al., 1991].

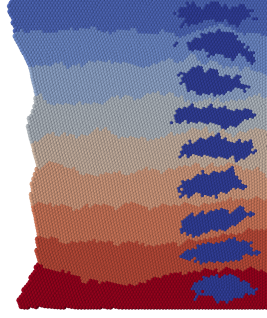
To set the compression parameter values, we note that it was shown by Sandersius and Newman [2008] that, for a 3D ScEM cell under small strains of less than 0.05, the timescale for equilibration is approximately 1 s. We found that these results also hold for the 2D model, as shown in Appendix A.3. We therefore choose a compression rate $C = 0.025 \text{ s}^{-1}$ and compress at 2 s intervals to allow the system to equilibrate. We carry out 10 successive compressions over a 20 s period. To obtain a balance between testing our model with different initial random seeds for the Gaussian noise and the initial construction of the cells and the limited resource availability of our shared HPC system, we carry out five instances of each simulation and consider the mean of each metric σ_{cell} and σ_{nuc} across the instances.

Figures 6.4(a)-(b) and Figures 6.4(c)-(d) show the initial and final states of the control case $K_{\text{nuc}} = 1.0$ and the test case $K_{\text{nuc}} = 10.0$ respectively. After 10 compressions the layer is compressed to 50% of its original width and we observe that all cells maintain their initial columnar shape, but their aspect ratio increases from 2 to approximately 8. The apical surface of the layer maintains a broadly flat morphology with the exception of the boundary cells which are deformed by the boundary walls. For the case $K_{\text{nuc}} = 1.0$ in Figure 6.4(b) we observe that the region labelled as the cell nucleus deforms with the cytoplasm to become elongated, and shows some disintegration as elements rearrange under compression, whereas for larger values of the nuclear stiffness shown in Figure 6.4(d) we see the nuclei maintains their integrity and form a buckled pattern.

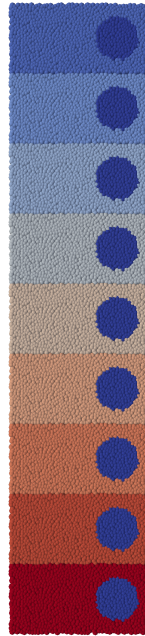
In Figure 6.5 we plot σ_{cell} and σ_{nuc} 1 s after each compression has been applied. By comparing Figure 6.5(a) with Figure 6.5(b) we see that for the control case $K_{\text{nuc}} = 1.0$ after 50% compression the standard deviation in the y component of the nucleus is similar to the standard deviation of the whole cell. However, for the test cases by comparing Figure 6.5(a) and Figure 6.5(b) we see that after a compression of approximately 35%



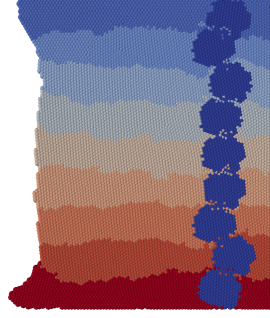
(a)



(b)

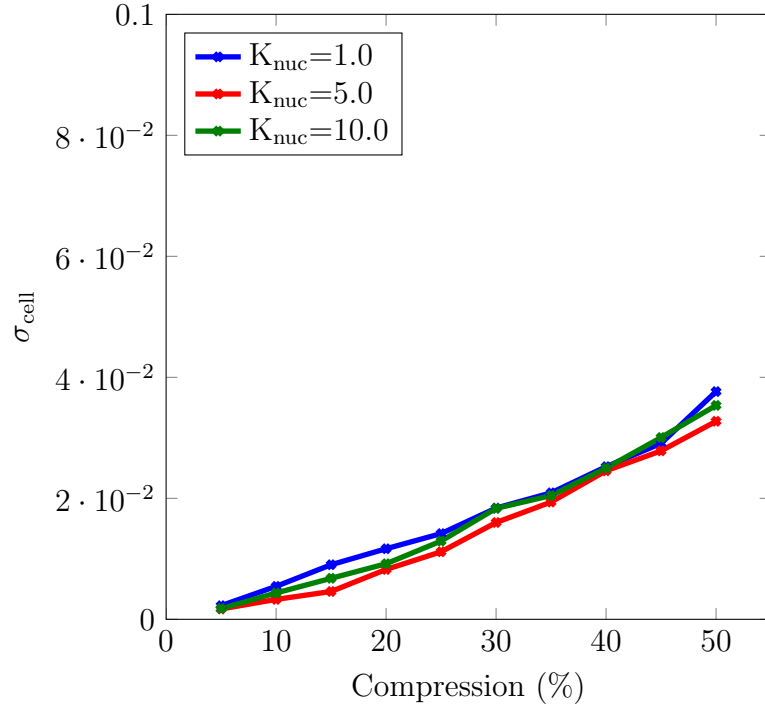


(c)

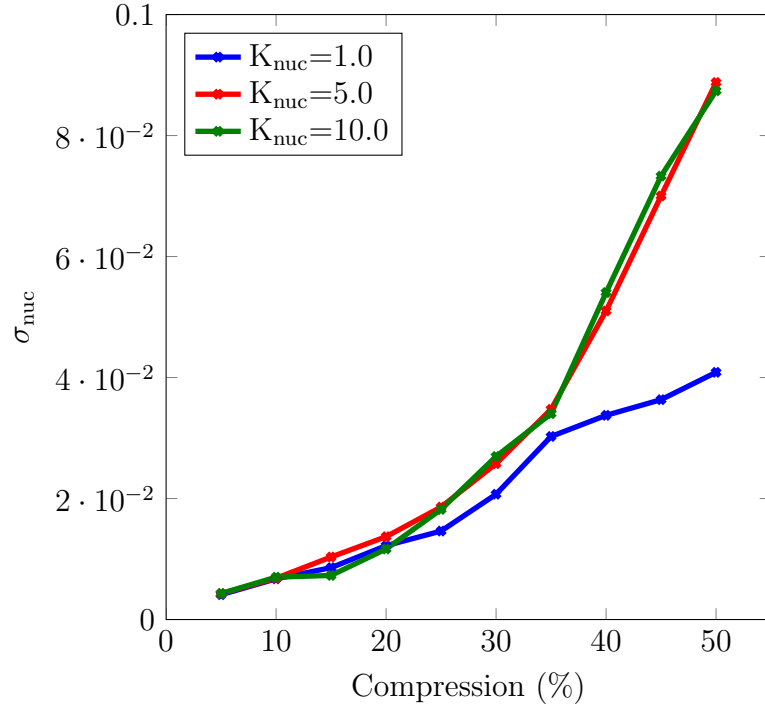


(d)

Figure 6.4: Initial and final states for cell layer compression experiments where (a)-(b) $K_{\text{nuc}} = 1.0$ and (c)-(d) $K_{\text{nuc}} = 10.0$. Cells are coloured arbitrarily, and their nuclei are coloured dark blue.



(a)



(b)

Figure 6.5: The standard deviation of the height of the centres of mass of: (a) each cell and (b) each cell nucleus for the cases where the relative nuclear stiffness is given by $K_{\text{nuc}} = 1.0$, $K_{\text{nuc}} = 5.0$, $K_{\text{nuc}} = 10.0$.

(14 s) of the layer that the deviation of the centre of mass of the nuclei increases sharply above both the control case and the deviation of the centre of mass of the whole cell. In Figure 6.4(d) this is observed as a buckled formation of the cell nuclei. This is in agreement with *in vitro* observations: the nucleus occupies approximately 65% of the cell width in our model cells, and we observe distinctive buckling of the nuclear structures as the layer is compressed to approximately 65% of its initial width. For the two test cases of $K_{\text{nuc}} = 5.0$ and $K_{\text{nuc}} = 10.0$, no difference is observed in the buckling dynamics of either the nuclei or the cells and we henceforth only consider the cases $K_{\text{nuc}} = 1.0$ and $K_{\text{nuc}} = 10.0$.

6.3.2 Effect of relative cell size

To assess how the relative size of stem and Paneth cells affects the buckling dynamics of cell nuclei, we begin by altering the relative sizes of stem cells in accordance with the data from Näthke and Appleton, and visible in Figures 6.2(a) and 6.2(b). Stem (S) cells are constructed to be half the width ($3.13\mu\text{m}$) of Paneth (P) cells and the stem cell nuclei are set to fill approximately 85% of the cell width with $R_{\text{nuc}} = 1.34\mu\text{m}$ and $\mathbf{r}_{\text{nuc}} = (0\mu\text{m}, -3.5\mu\text{m})$. Because the stem cells are smaller, we compress at a reduced rate $C = 0.015 \text{ s}^{-1}$ and compress at 2 s intervals over the same 20 s period. We initially consider an alternating pattern of stem and Paneth cells (SPSPSPSPS) with the initial and final conditions shown in Figure 6.6(a)-6.6(b).

Figure 6.7(a) (red and blue lines) shows σ_{cell} is unchanged by setting either $K_{\text{nuc}} = 1.0$ or $K_{\text{nuc}} = 10.0$, which is consistent with the results of the previous section. In Figure 6.7(b) we also see that for a stiffer nuclear region (red line) there is an increase in the buckling metric over the homogenous nuclear region at approximately 15% compression. This is consistent with our construction of the stem cells with their nuclei occupying approximately 85% of the cell width.

We further consider whether the pattern of the different types in the layer may have an effect on the buckling dynamics observed. We simulate a persistent pattern of stem cells at

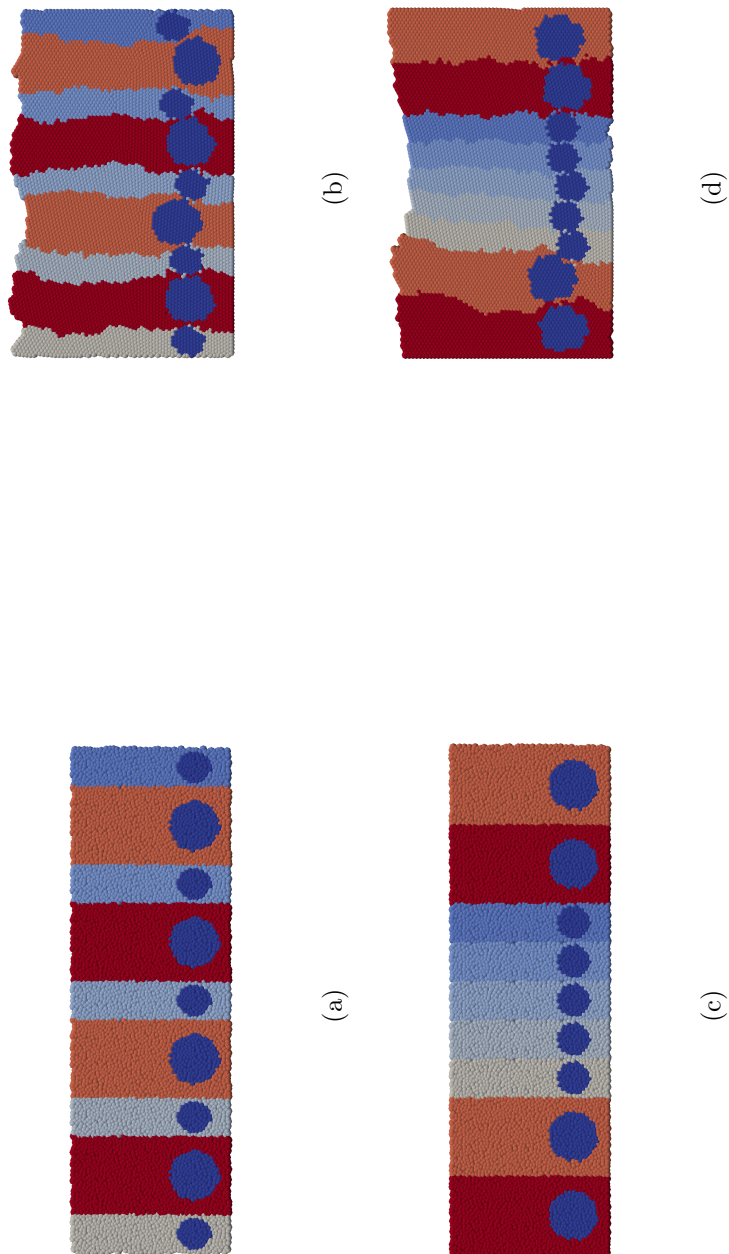
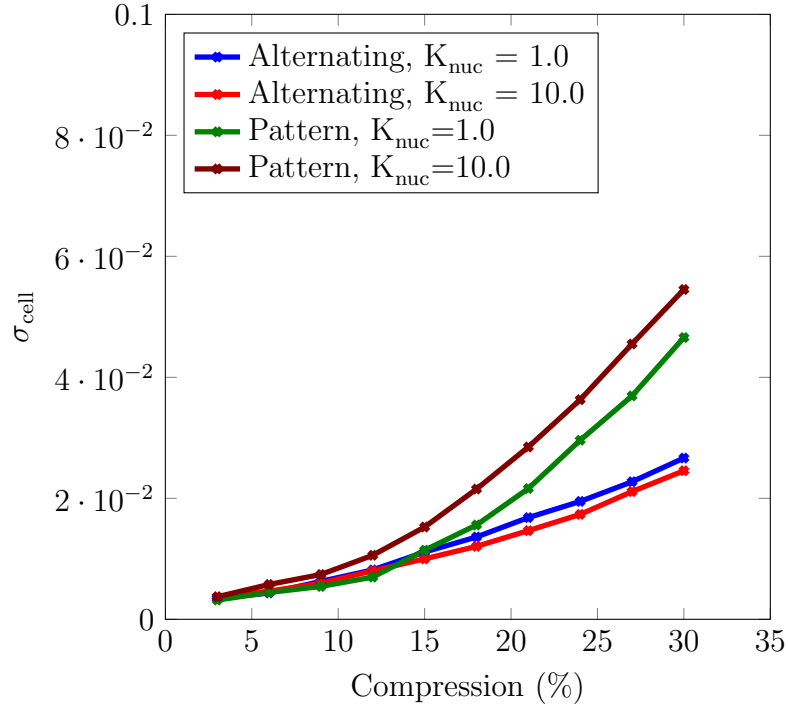
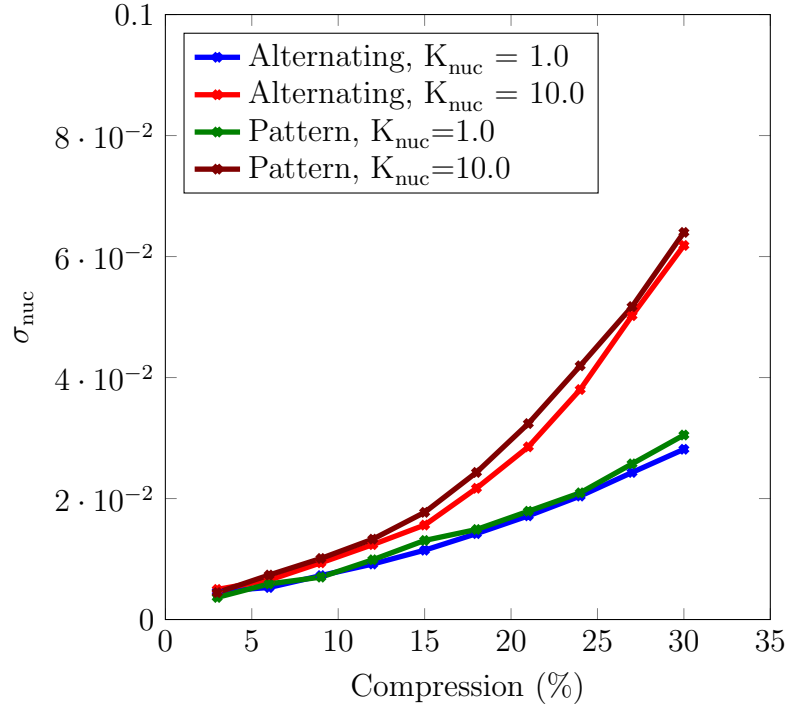


Figure 6.6: Initial and final states for cell layer compression experiments with (a)-(b) alternating Paneth and stem cells and (c)-(d) a persistent pattern of stem cells. Paneth cells are coloured red/orange and stem cells are coloured shades of blue. Cell nuclei are coloured dark blue.



(a)



(b)

Figure 6.7: The standard deviation of the height of the centres of mass of: (a) each cell and (b) each cell nucleus for the alternating pattern of cell size and the persistent pattern of stem cells.

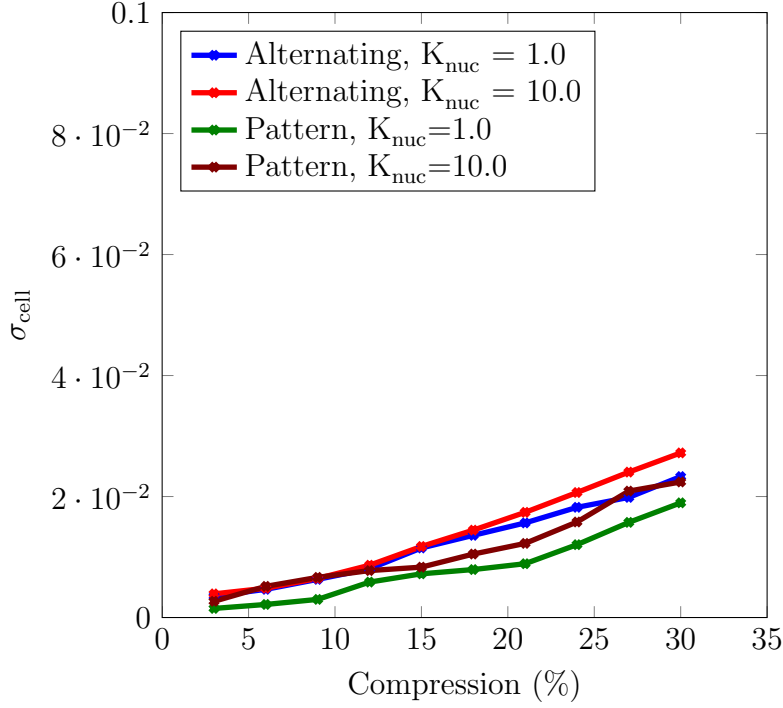


Figure 6.8: The standard deviation of the height of the centres of mass of each of the central 5 cells where cell size is varied. There is no significant deviation between the different patterns of cells, in contrast to Figure 6.7(a), suggesting that deformation of the boundary cells may account for this result.

the centre of the population (PPSSSSSP) as shown in Figure 6.6(c)-6.6(d). Figure 6.7(a) (green and maroon lines) again shows that little difference is seen between the cell level deviations with stiffer or homogeneous nuclear regions. However, it is noticeable that this pattern leads to a larger variance in the cell deviations over the alternating pattern. By considering the variation σ_{cell} for the central five cells, which we show in Figure 6.8, this difference is not observed, suggesting that this may be accounted for by the different cell sizes at the boundary and the only approximate scaling of the model parameters with element number. We discuss this in more detail in the next section. Figure 6.7(b) shows similar results for the persistent stem cells as for the alternating pattern, suggesting that simply the pattern of cell sizes is not sufficient to alter the point at which buckling of the nuclei is observed, or to increase the total amount of deflection as the compression is increased.

6.3.3 Effect of relative cell stiffness

While no data are available on the relative bulk stiffness of Paneth and stem cells, we are able to test the possible effects of variation in this property on the buckling of the cell nuclei using our model. As we have noted, the Paneth cell contains granular compartments containing anti-microbial peptides [Ganz, 2000], which may act to increase the cytoplasmic stiffness. By setting $K_i = 2.0$ for Paneth cells, we can test whether this may affect the buckling dynamics of the cell nuclei. As in the previous experiment we compress at a rate $C = 0.015 \text{ s}^{-1}$ and compress at 2 s intervals over a 20 s period.

In Figure 6.9 we visualise the initial and final condition for the alternating and patterned case, where the Paneth cell cytoplasm is set to be stiffer. Similarly to the previous experiments we observe buckling patterns of the nuclei, with no noticeable overall deflection of the cells. By plotting σ_{cell} and σ_{nuc} in Figure 6.10(a), we see again that no overall difference is seen in the deviations of the cell centres of mass, and that both the alternating and patterned cells behave similarly in this parameter regime. Furthermore, in Figure 6.10(b) we see that there is no difference in either the buckling point ($\approx 15\%$ compression), or the overall level of buckling between the different cell patterns and the previous experiment where we vary cell size alone.

6.3.4 Computational tractability

The results in this chapter have been made significantly more tractable by the work in Chapters 4 and 5. For the simulations of cell layer buckling we obtained a total speed-up of around 10 times using six processors from the original serial algorithm. This reduced compute time from over a week to around a day for a single simulation, enabling faster model development and the consideration of a greater number of hypotheses than would previously have been possible.

The speed-up is accounted for in approximately equal parts by speed-up from the serial algorithm by using more efficient data structures (Section 5.3), and the use of the parallel algorithm on six cores with approximately 50% efficiency. Transient compression of the

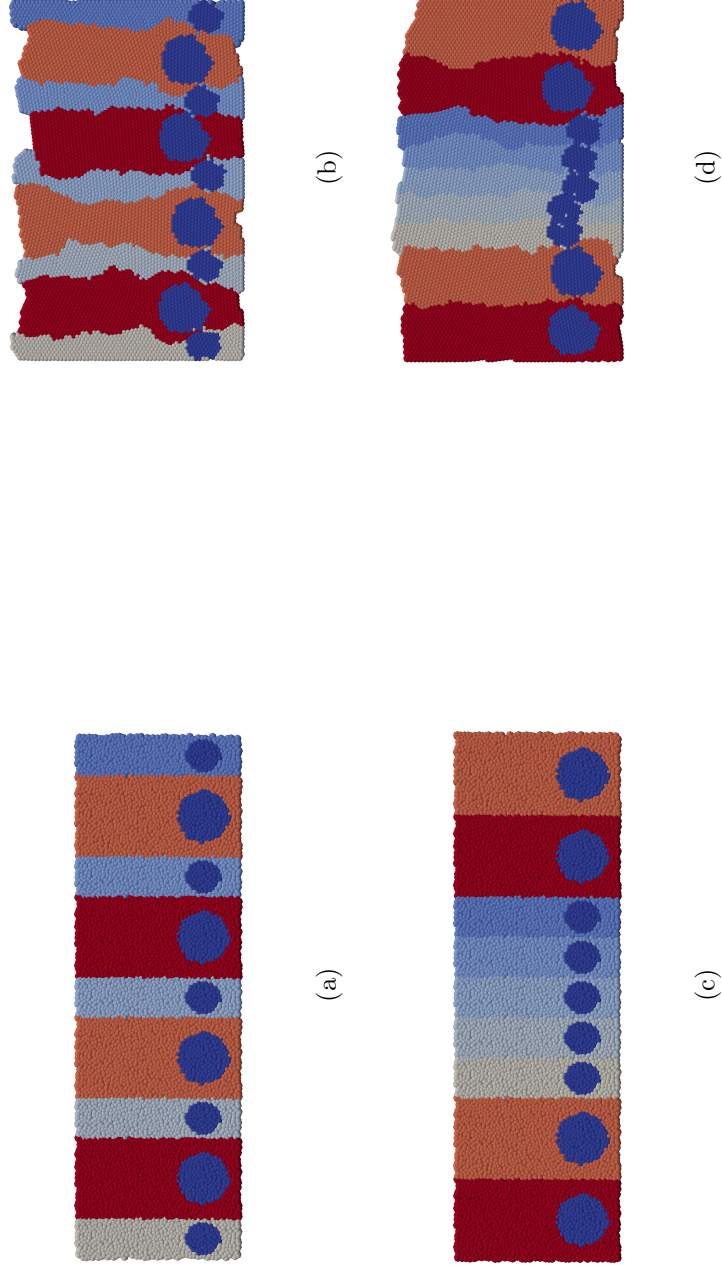
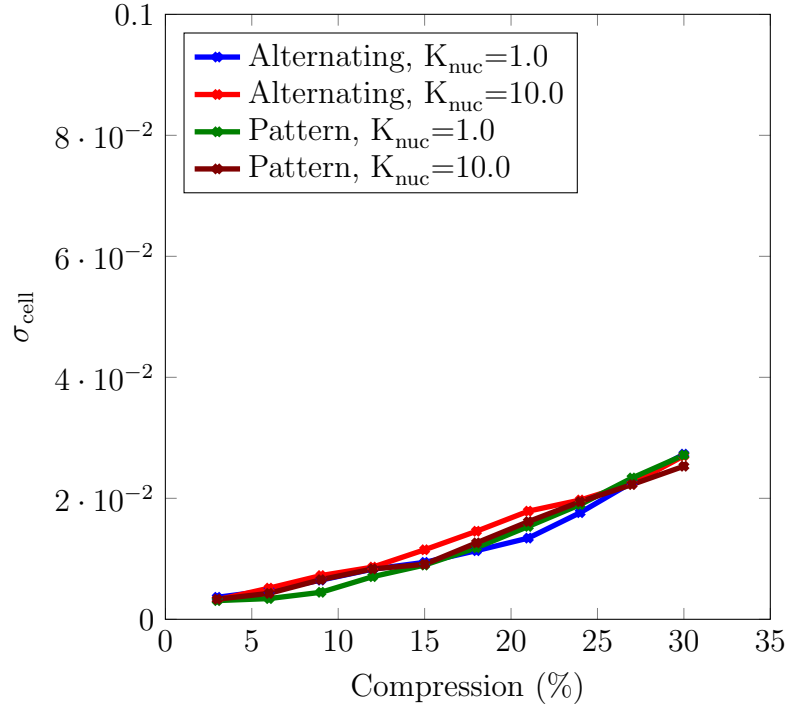
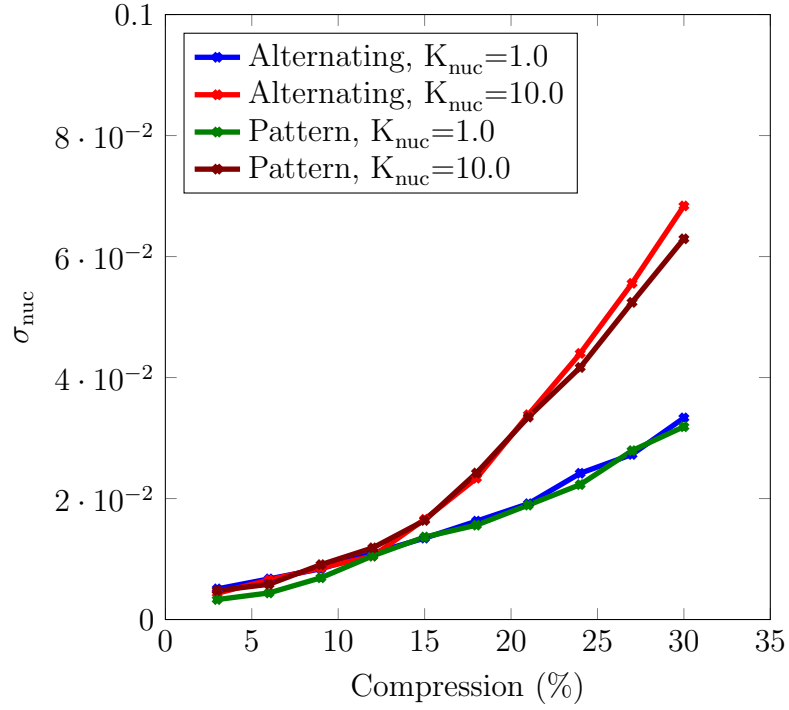


Figure 6.9: Initial and final states for cell layer compression experiments with $K_i = 2.0$ for Paneth cells with (a)-(b) an alternating pattern of stem and Paneth cells and (c)-(d) a persistent pattern of stem cells between two Paneth cells. Paneth cells are coloured red/orange and stem cells are coloured shades of blue. Cell nuclei are coloured dark blue.



(a)



(b)

Figure 6.10: The standard deviation of the centres of mass of: (a) each cell and (b) each cell nucleus for the alternating pattern of cells and the persistent pattern of stem cells where $K_i = 2.0$ for Paneth cells.

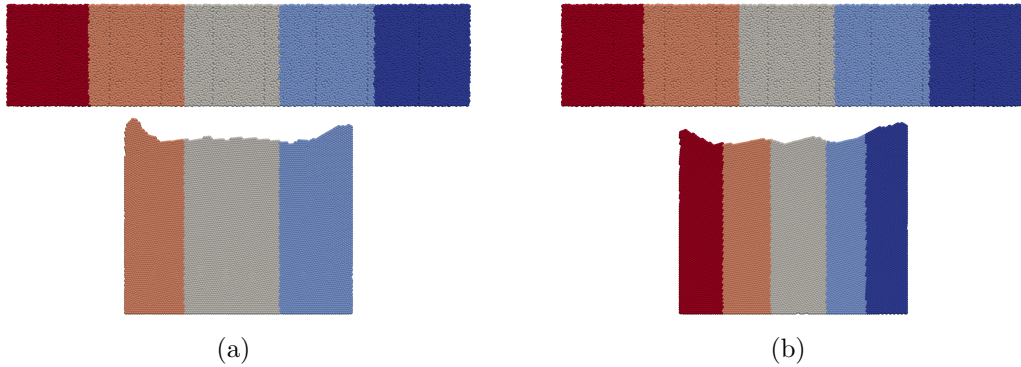


Figure 6.11: Dynamic load-balancing on five processors maintains an approximately even division of elements between processors in the compression experiments. Elements are coloured by the rank of their owning processor. (a) A simulation that does not use load-balancing begins with an even distribution of elements, but they are located on only three processors by the end of the simulation. (b) With dynamic load-balancing we maintain an approximately even division of elements.

layer led to load-imbalance, where all elements resided on only the central processors at the end of the simulation, as shown in Figure 6.11(a). By using the dynamic load-balance algorithm developed in Section 5.8 we were able to maintain an approximately even distribution of elements across all processors for the entire simulation, as shown in Figure 6.11(b).

For this application, future efficiencies may be gained by improving the serial algorithm. The calculation of the force in equation (6.3) is computationally expensive, and may be improved by creating a look-up table for $-(1/r)\partial V/\partial r$ and using linear interpolation to obtain estimates for the force magnitude. Furthermore, the integration scheme in Chaste for the equation of motion for each element uses the forward-Euler method. By using an implicit method or higher order scheme we may be able to increase the size of the time step Δt , and consequently improve the speed of a given simulation.

6.4 Discussion

In this chapter we have implemented the ScEM in Chaste and extended the parallel algorithm developed for cell centre models in Chapters 4 and 5 so that it may be used for the ScEM. We have proposed a new approach to modelling the instigation of crypt fission events using the ScEM. By using a biomechanically plausible model that allows a representation of subcellular biomechanical heterogeneity, we are able to suggest a possible trigger for fission events. The model predicts that a stiffer nuclear region in cells is vertically deflected when a layer of cells is laterally compressed, and that the larger size of nuclei relative to the cell width in stem cells observed *in vitro* leads this vertical nuclear deflection to occur under less compression than for a uniform row of Paneth type cells.

We constructed cells in our simulations to match the aspect ratio and relative nuclear size of cells observed *in vivo* in healthy, unbuckled murine crypts of the small intestine. We observe that a relatively large compression of 25%-35% of the initial layer width is required to observe deflection of the cell nuclei but this is reduced to approximately 15% when stem cells are introduced. This is in line with the *in vitro* observations that buckling events occur as a cell is compressed to the width of its nuclear diameter. In terms of *in vivo* crypt fission, these results are consistent with the fact that increased rates of fission are observed in cancerous tissue compared to healthy tissue. In healthy crypts of the small intestine, cell proliferation is balanced by migration of cells along the crypt-villus axis. However, in cancerous tissue, hyper-proliferation disrupts this balance and leads to crowding and compression of cells, which our simulations suggest may trigger an out-of-plane buckling of the cell nuclei.

We considered the further hypothesis, motivated by *in vitro* observations, that buckling occurs at sites where there are multiple neighbouring stem cells. Recreating this pattern in our model, we did not observe this behaviour. Clearly, as well as the passive role of the nucleus, other mechanisms that lead to crypt fission are at play and may explain this phenomenon. For example, the fact that it is the stem cells that divide may cause increased local compression as a result of the growth and division of multiple neighbouring

stem cells, causing a destabilisation of the monolayer structure of the epithelium at that point.

In all of our experiments we did not observe any significant buckling behaviour of the cells as a whole. While we would not expect buckling at the cell level to occur as a result of just nuclear deflections, a limitation of our model is the short-range nature of the interaction of the nucleus with the rest of the cell cytoplasm. In eukaryotic cells the nucleus is anchored into the cell by intermediate filaments [Maniotis et al., 1997] and so a deflection of the nucleus may have non-local effects not captured by our model. Furthermore, factors such as changes in cell-cell adhesion may play a role in epithelial layer buckling. For example, there is experimental evidence for the role of changes in adhesion at the apical surface of epithelial cells in buckling [Wang et al., 2012]. Furthermore, our model fails to account for the cell membrane, that encloses that cytoplasm and may offer greater resistance to cell deformation. The inclusion of these phenomena could form part of the future development of this modelling approach.

We noted in Section 6.3.2 that, although we scale model parameters to ensure that cells with different numbers of elements have the same bulk properties, in our simulations stem and Paneth cells behave differently at the interface of the boundary conditions and the cells. The approximate scaling of k_0 , the inter-element spring constant, assumes that cells are formed of an ordered network of springs, and that the network is of equal height to width. However, in our experiments the aspect ratio of cells is continuously changing as they are compressed, making this approximation less valid. Significant development of the model would be required to overcome this drawback. However, we do not believe it would significantly alter the main conclusions of our study, which are restricted to variation in the centres of mass of the cell nuclei, rather than the whole cell.

While our model only considers a flat underlying geometry, rather than the tightly curved base of the crypt, our results still have significance in the context of crypt fission. Particularly in cancerous adenoma, budding and fission events are observed to occur along the sides of the crypt, as well as at the base, as a result of the proliferative compartment

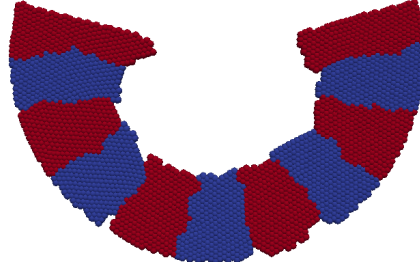


Figure 6.12: An example of a compressed layer of ScEM cells on a curved geometry, approximating the base of the crypt.

extending towards the crypt orifice [Wong et al., 2002]. Nevertheless, a further extension of our approach would be to carry out our simulations on a more realistic, curved underlying geometry at the base of the crypt. This could be implemented using a circular hard-wall boundary, as shown in Figure 6.12. Further consideration must be given to the approach to modelling compression throughout the layer using this approach, as a simple linear rescaling will not suffice. Applying a force to the elements at the extremity of the layer that is tangential to the surface causes large deformations to these cells.

Chapter 7

Discussion

Over the last few years a wide variety of individual-based cell population models have been developed for studying applications from tumour growth, the dynamics of epithelial layers and developmental processes. As these models become more detailed in both the geometric and mechanical representation of the cell and also integrate more available experimental data into the individual model of a cell, the need for increased computing power becomes more pressing. There has been a relative lack of development of open-source parallel algorithms to address this problem in the field of computational cell population modelling, and in particular for off-lattice models. In this thesis we have addressed this problem by developing a parallel algorithm for a cell centre model that we outlined in Chapter 3, which also naturally extends to a subcellular element model (ScEM). This has allowed us to study an example application using the Chaste library, beyond the scale of any previous simulation using the library. Furthermore, the generality of our approach and the fact that it is available as part of Chaste means that the benefit of our work is available to other scientists and is indeed already being used in D.Phil projects within the Computational Biology Group at the University of Oxford.

7.1 Parallelisation

In Chapter 4 we described our approach to spatial decomposition of data within a parallel simulation of a cell centre model. Motivated by the strictly near-neighbour mechanical and signalling interactions between cells in this model, we chose to group cells together on processors based on their spatial proximity. This approach had two principal benefits. First, we were able to minimise the volume of data that must be transferred between processors at each time step to accurately simulate the model. Were we to divide the data arbitrarily and equally between the processors, it would be necessary to implement global communication between all processors to calculate cell-cell interactions. In the case of the simulations considered, we found that communication costs were high in any case, and so this approach would be unfavourable. Second, our approach reduced the memory footprint of our simulation. Each processor need only store data on a subset of cells, namely those that it owns and a small number from the neighbouring processors. Consequently, this means that our parallel algorithm also increases the memory bound on the size of simulation that may be simulated. To further this end, we developed an efficient approach for constructing a large initial cell population from a data file, removing the requirement that all initial data must be replicated across all processors to generate a spatial decomposition.

We have overcome the implementation issue of communicating model cells that contain non-trivial data structures representing, for example the progress of the cell through its cell cycle. In Chaste, these data are stored in cell objects as member variables of an abstract class, meaning that we do not know *a priori* how to define the present state of a cell. Furthermore, the class hierarchy of cell data is constantly evolving and being extended. To overcome these issues we have used methods for class serialisation to create messages to send between processors. The benefits of this approach are that any additional models of the cell cycle, or additional data classes stored within a cell object that are developed in the future, can easily be handled using our parallel approach. Furthermore, cell-cell interactions can depend on any of the cell data, without the need to provide ad-

ditional messaging functionality. For example, the cell-cell interaction force class defined in Section 3.1 depends on the distance between cells, their radii and whether either cell is undergoing apoptosis. We are able to modify this class to include a dependence on its proliferative type, stored as a property in the cell object, without the need to amend the communication routines in our parallel implementation. The drawback of this approach is that, in general, more data are communicated than is absolutely necessary to simulate the model. Although we are able to reduce this cost by enforcing that all processors operate on a homogeneous system and by using a binary serialisation approach, communication still dominates the algorithm as the number of processors is increased.

We have achieved a significant reduction in the observed cost of communication by overlapping it with useful computational tasks that are independent of the data being communicated. In Section 5.6, for a fixed-size benchmark problem on a shared-memory supercomputer, we saw a doubling of the speed-up achieved on 32 processors. Moreover, there did not appear to be a saturation of speed-up at this point, suggesting that further speed-up would be possible were more processors available. Due to the supercomputer being a shared resource, it is not possible to gain regular access to more than 32 cores. However, on a distributed-memory architecture, where communication occurs through a slower network, speed-up is still severely limited by communication costs associated with transferring halo cells between neighbouring processors. This limits the possibility of our implementation tackling extremely large problems which have a memory footprint that is too large for a single shared-memory machine. To enable such a simulation, the next step would be to specialise the general communication methods that we have developed so that only the minimal amount of data is transferred. Due to the modular nature of the code we have developed, this would be possible by creating a child class of the cell population and specialising the communication algorithm to extract and send only the necessary data. The limitation of this approach is that it would require a different implementation for different types of simulation. Furthermore, the full communication method would still be required to faithfully reproduce cells that migrate between processors.

One of the drawbacks of the spatial decomposition approach is that it does not take account of the distribution of computational load between processors. We mitigated this problem for non-regular distributions of cells, or growing populations, by developing a simple diffusive load-balancing algorithm. Each processor attempts to minimise the difference between the number of cells it owns and the number of cells owned by its neighbours. This approach was shown to make the parallel algorithm profitable for use in a simulation of a growing monolayer of cells, where previously little speed-up was achieved. We also demonstrated that the algorithm is relatively insensitive to the parameter which defines how regularly the algorithm is iterated, meaning that although the ideal choice for any given simulation is dependent on the simulation dynamics, the parameter needn't be highly tuned to achieve near-optimal performance. A natural next step in the future development of this algorithm is to more accurately capture the actual load on a processor. For example, it is not just cells that lie in the local portion of the spatial domain that load the processor; the presence of data from neighbouring processors also leads to additional computation. Therefore an obvious next step would be to alter the function that defines load to include the number of halo cells, as well as local cells.

7.2 Application case study

In Chapter 6 we used our parallel algorithm to enable a computational study of a possible passive biomechanical trigger for out-of-plane deformation of the epithelial lining of the murine small intestine. Such a deformation may lead to fission of colorectal crypts. In close collaboration with experimentalists at the Näthke lab (University of Dundee) we investigated two hypotheses: first, whether the difference in stiffness of the cell nuclei could cause an out-of-plane deflection in a compressed layer of epithelial cells; and second, what effect the different cell types found in the base of the murine small intestine may have on this phenomenon.

In order to study the effect of subcellular biomechanical heterogeneity on the buckling

of a layer of epithelial cells under compression, we moved to a more detailed analogue of the cell centre model outlined in Chapter 3, the ScEM, where each cell is represented as a set of elements rather than a single point in space. As a result of the work in Chapters 4 and 5 the cost of simulating this model in Chaste was reduced by approximately an order of magnitude, enabling investigations of the two hypotheses to be made in reasonable time. While this case study does not include the detail of cell growth and division that our parallel algorithm is capable of handling, it does demonstrate the flexibility of our approach to the development of our algorithm, and the broader development of the Chaste library. With relatively little effort, we were able to extend the applicability of our work to the ScEM.

The results of our study suggest that cell nuclei in a sufficiently compressed epithelial layer will be deflected as a result of their increased stiffness. We found that when the cells were compressed to the width of their nuclei, our metric of buckling of the nuclei increases sharply above the control case. While this was not replicated at the cell level with an increase in buckling of the cell layer, we suggest this is due to our model not including the anchoring of the cell nucleus into the cell by intermediate filaments. Maniotis et al. [1997] saw the deformation of the cell boundary induce a deformation of the nucleus, suggesting that there are non-local effects that our model does not account for.

We also found that introducing stem- and Paneth-sized cells into our model, where the relative cell and nuclear sizes are calibrated in line with experimental observations, led to buckling under smaller compressions. This was as a result of the relatively larger size of nuclei in stem cells compared to Paneth cells. However, we found no evidence that the patterning of cells seen in fissioning crypts *in vivo*, where the buckling point occurs in a patch dominated by stem cells, increases the buckling metric or changes the point at which buckling occurs. Furthermore, we considered whether differences in the relative bulk stiffnesses of the whole cells may contribute to an increase in the buckling metric, or buckling under smaller compression. We found that increasing the bulk stiffness of the model Paneth cells did not significantly effect the result of the nuclear deflections.

This case study represents the first steps toward a detailed biomechanical model of the processes that drive crypt fission. Future development of this model within the Chaste library will be possible as a result of the parallel algorithm we have developed. This could include the non-local effects of the nuclear deflections as we have noted above, as well as incorporating the role of the cell membrane in the cellular resistance to deformation. Moving to a realistic crypt geometry would also further aid qualitative comparison of our results with the experimental images of *in vivo* murine crypts. The *in vitro* organoid models of crypt-villus structures developed by Sato et al. [2009] also offer a framework for qualitative comparison and calibration of our model in the future.

From the perspective of improving the computational performance of our algorithm for this specific application, in addition to the ideas presented for specialisation of the parallel algorithm, we also suggest that further efficiency may be gained by approximating the potential function for the inter-element force using a look-up table, as well using a higher-order method for integrating the equation of motion for each cell (equation 3.2) or subcellular element (equation 6.1), enabling a smaller time step to be used. In concert these developments may allow a wider range of parameters to be explored, and a greater number of instances of each simulation to be run to fully understand the effect of the random initial condition of elements within a cell, and the Gaussian noise term representing thermal fluctuations in each element.

In conclusion, this thesis has developed and implemented a parallel algorithm for a general class of off-lattice cell population models that is flexible enough to be applied to many different modelling applications, and may be easily extended or specialised for different studies. We have shown its applicability and extensibility by suggesting a new possible mechanism that initiates epithelial layer buckling in the murine small intestine that explains experimentally observed phenomena.

Appendix A

Scaling the parameters of the subcellular element model

In this appendix we derive scalings for the 2D ScEM, and refer the reader to Sandersius and Newman [2008] where the 3D case is considered.

A.1 Scaling the equilibrium distance r_{eq}

The area of a circular cell with natural radius R_{cell} is given by πR_{cell}^2 . Considering the plane packing of N circles of radius $r_{\text{eq}}/2$ within the cell area yields the relation

$$p_2 \pi R_{\text{cell}}^2 = N \pi \left(\frac{r_{\text{eq}}}{2} \right)^2,$$

where p_2 is the packing density of circles. Hence we obtain

$$r_{\text{eq}} = 2R_{\text{cell}} \left(\frac{p_2}{N} \right)^{\frac{1}{2}}.$$

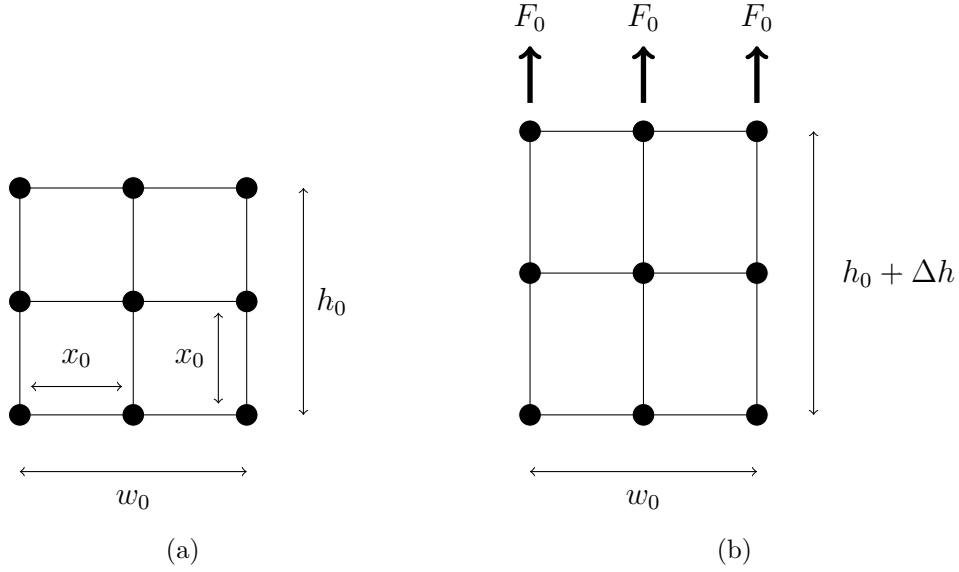


Figure A.1: The ordered network of elements considered to scale k_0 . Applying a constant pressure to the upper surface induces a strain in the network. The ratio of stress to strain may be kept constant by scaling k_0 depending on the number of elements in the system.

A.2 Scaling the elastic potential

To obtain a scaling for the constant u_0 we follow Sandersius and Newman [2008] by first linearising the derivative of the potential in equation (6.2) to obtain a linear spring constant which approximates the force for small deviations from equilibrium. By considering the Taylor expansion of $V'(r)$ about $r = r_{\text{eq}}$ we obtain

$$V'(r) = \frac{8\rho^2 u_0}{r_{\text{eq}}^2} (r - r_{\text{eq}}) + O(|r - r_{\text{eq}}|^2),$$

giving a linear spring constant

$$k_0 = \frac{8\rho^2 u_0}{r_{\text{eq}}^2}.$$

To obtain an approximate scaling for k_0 , we consider an ordered network of elements with macroscopic dimension $w_0 \times h_0$ composed of $n_w \times n_h$ elements with equilibrium separation x_0 , as shown in Figure A.1(a). The number of elements is given by $N = n_w n_h$ and we

obtain the following relations:

$$w_0 = x_0(n_w - 1), \quad \text{and} \quad h_0 = x_0(n_h - 1).$$

By applying a constant force F_0 to the top boundary elements we induce a linear deformation of the system, as shown in Figure A.1(b). The stress on the top boundary is given by

$$\sigma = \frac{n_w F_0}{w_0}, \tag{A.1}$$

and the strain is given by $\epsilon = \Delta h/h_0$. By approximating each element-element interaction by a linear spring with constant k_0 and applying Hooke's law to each column of springs we obtain

$$F_0 = \frac{k_0}{n_h - 1} \Delta h. \tag{A.2}$$

Combining equations (A.1) and (A.2) we obtain

$$\frac{\sigma}{\epsilon} = \frac{h_0 n_w}{w_0 (n_h - 1)} k_0.$$

Thus, to ensure that the bulk stiffness of the cell (given by σ/ϵ) is independent of N , we scale k_0 , and hence u_0 , by $(1 - \lambda N^{\frac{1}{2}})$ where λ is a tuning parameter introduced by Sandersius and Newman [2008] to adjust for the fact that this scaling is only approximate as elements do not generally form the simple ordered network considered here.

A.3 2D creep stress experiments

To assess the viscous time scale of the model cells using our implementation of the ScEM in Chaste, we carry out a 2D analogue of the creep stress experiment of Sandersius and Newman [2008]. Following the protocol of Sandersius and Newman [2008], we place a circular ScEM cell of radius $R_{\text{cell}} = 5\mu\text{m}$ consisting of 256 elements between two ‘plates’,

Parameter	Description	Value	Reference
N	Number of elements per cell	256	–
r_c	Force cut-off distance	$0.1 (\approx 2r_{\text{eq}})$	–
η	Damping constant	$10^{-9} \text{ N s } \mu\text{m}^{-1}$	Sandersius and Newman [2008]
k_0	Effective spring constant	$10^{-9} \text{ N } \mu\text{m}^{-1}$	–
ρ	Elastic constant	2.0	Sandersius and Newman [2008]
λ	Elastic scaling parameter	0.5	Sandersius and Newman [2008]
D	Diffusion constant	0	–
R_{cell}	Radius of isolated cell	$5\mu\text{m}$	–
p_2	Packing density of circles	$\pi/2\sqrt{3}$	Sandersius et al. [2011]
Δt	Integration time-step	10^{-3}s	–

Table A.1: Parameter values for the 2D creep strain experiments. The values η and k_0 are the unscaled values.

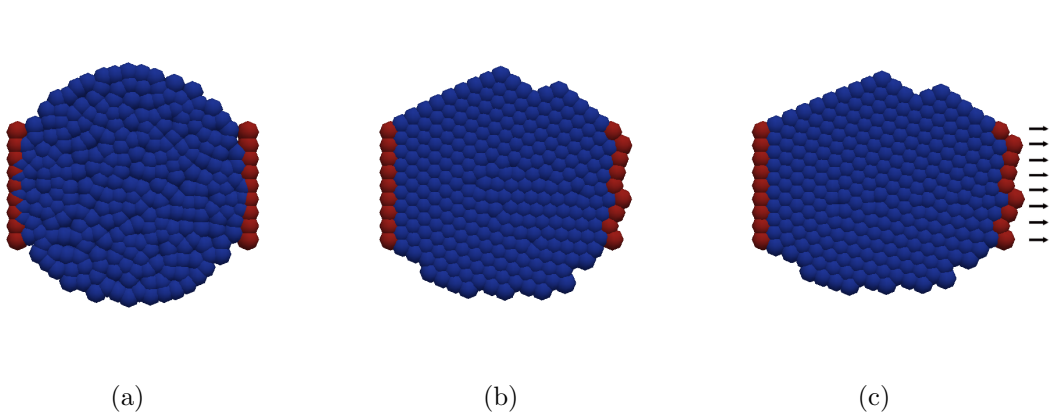


Figure A.2: Set up for the 2D rheology experiments. (a) Initial condition of cell (blue) and rheology plates (red). (b) After relaxation the rheology plate adheres to the cell. (c) Cell extended by pressure (shown by arrows) applied to one rheology plate, while other plate is held fixed.

also composed of ScEM elements. The plates are approximately $5\mu\text{m}$ long and contain nine equally spaced elements, separated by a distance r_{eq} . As shown in Figure A.2(a), the plates are oriented parallel to each other and tangential to the cell. Elements of the rheology plates interact with cell elements through the same force laws as those for two cell elements, but the value K_{diff} for interaction between rheology plate elements and cell elements is set to 1.0 to ensure that the plate stays attached to the cell. Elements of one plate are held fixed, and elements of the other plate are constrained to move perpendicular to the initial extension of the plate.

This system is allowed to relax for 5 s where the cell reaches equilibrium and the free

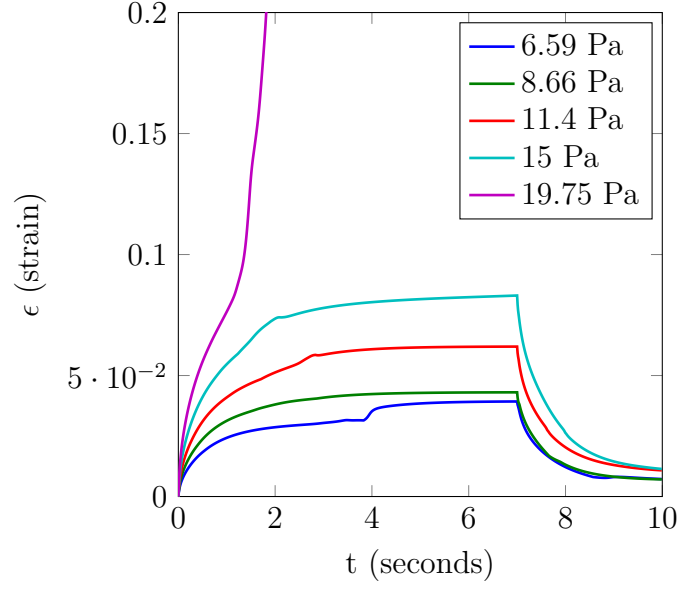


Figure A.3: Strain (ϵ) over time for 2D rheology experiments carried out in Chaste for different values of applied stress. The cell is stressed for 7 s before being released.

rheology plate adheres to the cell surface, shown in Figure A.2(b). A stress is applied to the free plate, which is equally distributed between the elements of the plate and perpendicular to the plates initial extension. This induces a strain in the cell as shown in Figure A.2(c), which we measure as the linear extension of the cell from equilibrium divided by its equilibrium extension parallel to the applied force. After 7 s the applied pressure is released and the cell allowed to relax.

Figure A.3 plots strain against time from when the extension force is applied to the rheology plate. We see close agreement with the results in 3D from Sandersius and Newman [2008], where for small strains the cell reaches extended equilibrium after approximately 1 s. On releasing the applied force the cell returns to an equilibrium state after a further 1 s. For large strains beyond approximately 0.075 we see that the cell breaks apart. For small stresses, we note that the cell does not completely return to its reference strain ($\epsilon = 0$) when the force is released. This is due to the rearrangement of the neighbourhood topology of elements as a result of the applied stress.

Bibliography

- S. Adra, T. Sun, S. MacNeil, M. Holcombe, and R. Smallwood. Development of a three dimensional multiscale computational model of the human epidermis. *PLoS ONE*, 5 (1):e8511, 2010.
- Y. Afshar, F. Schmid, A. Pishavar, and S. Worley. Exploiting seeding of random number generators for efficient domain decomposition parallelization of dissipative particle dynamics. *Comput. Phys. Commun.*, 184(4):1119 – 1128, 2013.
- T. Alarcón, H.M. Byrne, and P.K. Maini. A cellular automaton model for tumour growth in inhomogeneous environment. *J. Theor. Biol.*, 225:257–274, 2003.
- B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. *Molecular Biology of the Cell*. Garland Science, 4th edition, 2002.
- S. Alberts, M.K. Keenan, R.M. D’Souza, and G. An. Data-parallel techniques for simulating a mega-scale agent-based model of systemic inflammatory response syndrome on graphics processing units. *Simulation*, 88:895–907, 2011.
- W. de Back, J. Starruss, and A. Deutsch. Morpheus: a novel modeling environment for multicellular systems biology. 2012. BIOTEC Forum: Bioinformatics and Computational Biology, Dresden.
- D.A. Bader and S. Aluru. High-performance computational biology. *Parallel Comput.*, 34:613 – 615, 2008.
- S. Balay, J. Brown, K. Buschelman, W.D. Gropp, D. Kaushik, M.G. Knepley, L.C. McInnes, B.F. Smith, and H. Zhang. PETSc web page, 2011. <http://www.mcs.anl.gov/petsc>.
- N. Barker, J.H. van Es, J. Kuipers, P. Kujala, M. van den Born, M. Cozijnsen, A. Haegebarth, J. Korving, H. Begthel, P.J. Peters, and H. Clevers. Identification of stem cells in small intestine and colon by marker gene Lgr5. *Nature*, 449(7165):1003–U1, 2007.
- M. Basan, J.-F. Joanny, J. Prost, and T. Risler. Undulation instability of epithelial tissues. *Phys. Rev. Lett.*, 106:158101, 2011.
- M. Ben Amar and F. Jia. Anisotropic growth shapes intestinal tissues during embryogenesis. *Proc. Natl. Acad. Sci. USA*, 110:10525–10530, 2013.

-
- H.J.C. Berendsen, D. van der Spoel, and R. van Drunen. GROMACS: A message-passing parallel molecular dynamics implementation. *Comput. Phys. Commun.*, 91:43 – 56, 1995.
- M.O Bernabeu, J. Southern, N. Wilson, P. Strazdins, J. Cooper, and J. Pitt-Francis. Chaste: a case study of parallelisation of an open source finite-element solver with applications to computational cardiac electrophysiology simulation. *Int. J. High Perform. C.*, In Press 2013.
- M. Bjerknes and H. Cheng. The stem-cell zone of the small intestinal epithelium. 1. Evidence from Paneth cells in the adult mouse. *Am J. Anat.*, 160(1):51–63, 1981.
- M. Brittan and N.A. Wright. The gastrointestinal stem cell. *Cell Prolif.*, 37(1):35–53, 2004.
- G.W. Brodland and H.H. Chen. The mechanics of heterotypic cell aggregates: Insights from computer simulations. *J. Biomech. Eng.-T. ASME*, 122(4):402–407, 2000a.
- G.W. Brodland and H.H. Chen. The mechanics of cell sorting and envelopment. *J. Biomech.*, 33(7):845–851, 2000b.
- G.W. Brodland and J.H. Veldhuis. Computer simulations of mitosis and interdependencies between mitosis orientation, cell shape and epithelia reshaping. *J. Biomech.*, 35(5):673–681, 2002.
- P. Buske, J. Galle, N. Barker, G. Aust, H. Clevers, and M. Loeffler. A comprehensive model of the spatio-temporal stem cell and tissue organisation in the intestinal crypt. *PLoS Comput. Biol.*, 7:e1001045, 2011.
- P. Buske, J. Przybilla, M. Loeffler, N. Sachs, T. Sato, H. Clevers, and J. Galle. On the biomechanics of stem cell niche formation in the gut - modelling growing organoids. *FEBS J.*, 279(18):3475–3487, 2012.
- H. Byrne and D. Drasdo. Individual-based and continuum models of growing cell populations: a comparison. *J. Math. Biol.*, 58:657–687, 2009.
- C++11. Information technology – Programming languages – C++. (ISO/IEC 14882), 2011.
- N. Caille, O. Thoumine, Y. Tardy, and J.J. Meister. Contribution of the nucleus to the mechanical properties of endothelial cells. *J. Biomech.*, 35(2):177–187, 2002.
- A.B. Cairnie and B.H. Millen. Fission of the crypts in the small intestine of the irradiated mouse. *Cell Prolif.*, 8(2):189–196, 1975.
- N. Chen, J.A. Glazier, J.A. Izaguirre, and M.S. Alber. A parallel implementation of the Cellular Potts Model for simulation of cell-based morphogenesis. *Comput. Phys. Commun.*, 176:670–681, 2007.
- S. Christley, B. Lee, X. Dai, and Q. Nie. Integrative multicellular biological modeling: A case study of 3D epidermal development using GPU algorithms. *BMC Syst. Biol.*, 4: 107, 2010.

-
- Y.S. Chu, S. Dufour, J.P. Thiery, E. Perez, and F. Pincet. Johnson-Kendall-Roberts theory applied to living cells. *Phys. Rev. Lett.*, 94:028102+, 2005.
- S. Coakley, S. Chin, D. Worth, and C. Greenough. FLAME web page, 2013. <http://www.flame.ac.uk/>.
- S. Coakley, M. Gheorghe, M. Holcombe, S. Chin, D. Worth, and C. Greenough. Exploitation of high performance computing in the FLAME agent-based simulation framework. pages 538–545, 2012.
- J.R. Collier, N.A.M. Monk, P.K. Maini, and J.H. Lewis. Pattern formation by lateral inhibition with feedback: A mathematical model of Delta-Notch intercellular signalling. *J. Theor. Biol.*, 183(4):429–446, 1996.
- A. Corradi, L. Leonardi, and F. Zambonelli. Diffusive load-balancing policies for dynamic applications. *Concurrency, IEEE*, 7(1):22–31, 1999.
- G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. *J. Parallel Distr. Com.*, 7:279–301, 1989.
- J.C. Dallon and H.G. Othmer. How cellular movement determines the collective force generated by the *Dictyostelium discoideum* slug. *J. Theor. Biol.*, 231(2):203–222, 2004.
- G. D’Antonio, P. Macklin, and L. Preziosi. An agent-based model for elasto-plastic mechanical interactions between cells, basement membrane and extracellular matrix. *Math. Biosci. Eng.*, 10:75–101, 2013.
- T. Darden, D. York, and L. Pedersen. Particle mesh Ewald: An $n \log(n)$ method for Ewald sums in large systems. *J. Chem. Phys.*, 98(12):10089–10092, 1993.
- Y. Davit, J.M. Osborne, H.M. Byrne, D. Gavaghan, and J. Pitt-Francis. Validity of the Cauchy-Born rule applied to discrete cellular-scale models of biological tissues. *Phys. Rev. E*, 87(4):042724, 2013.
- C. Dong, R. Skalak, and K.L.P. Sung. Cytoplasmic rheology of passive neutrophils. *Biorheology*, 28(6):557–567, 1991.
- D. Drasdo. Buckling instabilities of one-layered growing tissues. *Phys. Rev. Lett.*, 84: 4244–4247, 2000.
- D. Drasdo and S. Höehme. Modeling the impact of granular embedding media, and pulling versus pushing cells on growing cell clones. *New. J. Phys.*, 14:055025, 2012.
- D. Drasdo and S. Höhme. A single-cell-based model of tumor growth *in vitro*: monolayers and spheroids. *Phys. Biol.*, 2:133–147, 2005.
- D. Drasdo and M. Loeffler. Individual-based models to growth and folding in one-layered tissues: Intestinal crypts and early development. *Nonlinear Anal.-Theor.*, 47:245–256, 2001.
- D. Drasdo, R. Kree, and J.S. McCaskill. Monte-carlo approach to tissue-cell populations. *Phys. Rev. E.*, 52(6, B):6635–6657, 1995.

-
- W. Dückting and T.H. Vogelsaenger. Recent progress in modelling and simulation of three-dimensional tumor growth and treatment. *Biosystems*, 18(1):79 – 91, 1985.
- S-J. Dunn, P.L. Appleton, S.A. Nelson, I.S. Näthke, D.J. Gavaghan, and J.M. Osborne. A two-dimensional model of the colonic crypt accounting for the role of the basement membrane and pericryptal fibroblast sheath. *PLoS Comput. Biol.*, 8(5):e1002515, 2012a.
- S-J. Dunn, A.G. Fletcher, S.J. Chapman, D.J. Gavaghan, and J.M. Osborne. Modelling the role of the basement membrane beneath a growing epithelial monolayer. *J. Theor. Biol.*, 298(0):82 – 91, 2012b.
- C.M. Edwards and S.J. Chapman. Biomechanical modelling of colorectal crypt budding and fission. *Bull. Math. Biol.*, 69(6):1927–1942, 2007.
- G.B. Ermentrout and L. Edelstein-Keshet. Cellular automata approaches to biological modeling. *J. Theor. Biol.*, 160(1):97–133, 1993.
- R. Farhadifar, J-C. Roper, B. Algouy, S. Eaton, and F. Juelicher. The influence of cell mechanics, cell-cell interactions, and proliferation on epithelial packing. *Curr. Biol.*, 17(24):2095–2104, 2007.
- J.-L. Fattebert, D.F. Richards, and J.N. Glosli. Dynamic load balancing algorithm for molecular dynamics based on Voronoi cells domain decompositions. *Comput. Phys. Commun.*, 183(12):2608–2615, 2012.
- G.P. Figueredo, T.V. Joshi, J.M. Osborne, H.M. Byrne, and M.R. Owen. On-lattice agent-based simulation of populations of cells within the open-source Chaste framework. *Interface Focus*, 3(2):20120081, 2013.
- D. Fincham. Parallel computers and molecular simulation. *Mol. Simulat.*, 1:1–45, 1987.
- A.G. Fletcher, C.J.W. Breward, and S.J. Chapman. Mathematical modeling of monoclonal conversion in the colonic crypt. *J. Theor. Biol.*, 300:118–133, 2012.
- J.A. Fozard, H.M. Byrne, O.E. Jensen, and J.R. King. Continuum approximations of individual-based models for epithelial monolayers. *Math. Med. Biol.*, 27(1):39–74, 2010.
- F. Frascoli, B.D. Hughes, M.H. Zaman, and K.A. Landman. A computational model for collective cellular motion in three dimensions: General framework and case study for cell pair dynamics. *PLoS ONE*, 8(3):e59249, 2013.
- J.P. Freyer and R.M. Sutherland. Regulation of growth saturation and development of necrosis in EMT6/Ro multicellular spheroids by the glucose and oxygen supply. *Cancer Res.*, 46:3504–3512, 1986.
- S.M. Frisch and H. Francis. Disruption of epithelial cell-matrix interactions induces apoptosis. *J. Cell Biol.*, 124(4):619–626, 1994.
- J. Galle, M. Loeffler, and D. Drasdo. Modeling the effect of deregulated proliferation and apoptosis on the growth dynamics of epithelial cell populations *in vitro*. *Biophys. J.*, 88:62–75, 2005.

-
- E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1st edition, 1994.
- T. Ganz. Paneth cells - guardians of the gut cell hatchery. *Nat. Immunol.*, 1(2):99–100, 2000.
- J.A. Glazier and F. Graner. Simulation of the differential adhesion driven rearrangement of biological cells. *Phy. Rev. E*, 47:2128–2154, 1993.
- N. Goel, R. Campbell, R. Gordon, R. Rosen, H. Martinez, and M. Yas. Self-sorting of isotropic cells. *J. Theor. Biol.*, 28:423–468, 1970.
- F. Graner and J.A. Glazier. Simulation of biological cell sorting using a two-dimensional extended Potts model. *Phys. Rev. Lett.*, 69:2013–2016, 1992.
- L.C. Greaves, S.L. Preston, P.J. Tadrous, R.W. Taylor, M.J. Barron, D. Oukrif, S.J. Leedham, M. Deheragoda, P. Sasieni, M.R. Novelli, J.A.Z. Jankowski, D.M. Turnbull, N.A. Wright, and S.A.C. McDonald. Mitochondrial DNA mutations are established in human colonic stem cells, and mutated clones expand by crypt fission. *P. Natl. Acad. Sci. USA*, 103:714–719, 2006.
- W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: portable parallel programming with the Message-Passing Interface*. The MIT Press, 2nd edition, 1999.
- F. Guilak, J.R. Tedrow, and R. Burgkart. Viscoelastic properties of the cell nucleus. *Biochem. Bioph. Res. Co.*, 269(3):781–786, 2000.
- E. Hannezo, J. Prost, and J.-F. Joanny. Instabilities of monolayered epithelia: shape and structure of villi and crypts. *Phys. Rev. Lett.*, 107:078104, 2011.
- A. Henderson. ParaView Guide, A Parallel Visualization Application. *Kitware Inc.*, 2007.
- B. Hess, C. Kutzner, D. van der Spoel, and E. Lindahl. GROMACS 4: Algorithms for highly efficient, load-balanced, and scalable molecular simulation. *J. Chem. Theory Comput.*, 4:435–447, 2008.
- S.D. Hester, J.M. Belmonte, J.S. Gens, S.G. Clendenon, and J.A. Glazier. A multi-cell, multi-scale model of vertebrate segmentation and somite formation. *PLoS Comput. Biol.*, 7(10):e1002155, 2011.
- S. Höehme and D. Drasdo. A cell-based simulation software for multi-cellular systems. *Bioinformatics*, 26:2641–2642, 2010a.
- S. Höehme and D. Drasdo. Biomechanical and nutrient controls in the growth of mammalian cell populations. *Math. Popul. Stud.*, 17:166–187, 2010b.
- M. Holcombe et al. Modelling complex biological systems using an agent-based approach. *Integr. Biol.*, 4(1):53–64, 2012.
- H. Honda. Description of cellular patterns by Dirichlet domains: the two-dimensional case. *J. Theor. Biol.*, 72:523–543, 1978.

-
- H. Honda, M. Tanemura, and T. Nagai. A three-dimensional vertex dynamics cell model of space-filling polyhedra simulating cell behavior in a cell aggregate. *J. Theor. Biol.*, 226(4):439–453, 2004.
- C. Huang, O. Lawlor, and L.V. Kalé. Adaptive MPI. In *Proceedings of the 16th International Workshop on Languages and Compilers for Parallel Computing (LCPC 2003)*, LNCS 2958, pages 306–322, October 2003.
- J.A. Izaguirre, R. Chaturvedi, C. Huang, T. Cickovski, J. Coffland, G. Thomas, G. Forgacs, M. Alber, G. Hentschel, S.A. Newman, and J.A. Glazier. CompuCell, a multi-model framework for simulation of morphogenesis. *Bioinformatics*, 20:1129–1137, 2004.
- G.W. Jones and S.J. Chapman. Modeling growth in biological materials. *SIAM Rev.*, 54(1):52–118, 2012.
- B. Jung, H. Lenhof, P. Müller, and C. Rüb. Simulating synthetic polymer chains in parallel. *Future. Gener. Comp. Sy.*, 16:513–522, 2000.
- L.V. Kale and S. Krishnan. CHARM++: A Portable Concurrent Object Oriented System Based on C++. In *Proceedings of OOPSLA 93*, pages 91–108. ACM Press, September 1993.
- B. Karlsson. *Beyond the C++ Standard Library: An Introduction to Boost*. Addison-Wesley Professional, 1st edition, 2005.
- Y. Kim, M.A. Stolarska, and H.G. Othmer. A hybrid model for tumor spheroid growth *in vitro* I: theoretical development and early results. *Math. Mod. Meth. Appl. Sci.*, 17:1773–1798, 2007.
- L.A. Lardon, B.V. Merkey, S. Martins, A. Dötsch, C. Picoreanu, J-U. Kreft, and B.F. Smets. iDynoMiCS: next-generation individual-based modelling of biofilms. *Environ. Microbiol.*, 13(9):2416–2434, 2011.
- W.H. Lewis. Mechanics of invagination. *Anat. Rec.*, 97(2):139–156, 1947.
- X. Li, A.K. Upadhyay, A.J. Bullock, T. Dicolandrea, J. Xu, R.L. Binder, M.K. Robinson, D.R. Finlay, K.J. Mills, C.C. Bascom, C.K. Kelling, R.J. Isfort, J.W. Haycock, S. MacNeil, and R.H. Smallwood. Skin stem cell hypotheses and long term clone survival - explored using agent-based modelling. *Sci. Rep.*, 3(1904), 2013.
- W. Liu, B. Schmidt, G. Voss, and W. Müller-Wittig. Accelerating molecular dynamics simulations using graphics processing units with CUDA. *Comput. Phys. Commun.*, 179:634–641, 2008.
- R. Lozano et al. Global and regional mortality from 235 causes of death for 20 age groups in 1990 and 2010: a systematic analysis for the Global Burden of Disease Study 2010. *Lancet*, 380(9859):2095 – 2128, 2013.
- P.M. Lushnikov, N. Chen, and M. Alber. Macroscopic dynamics of biological cells interacting via chemotaxis and direct contact. *Phys. Rev. E*, 78(6, 1):061904, 2008.

-
- P. Macklin, M.E. Edgerton, A.M. Thompson, and V. Cristini. Patient-calibrated agent-based modelling of ductal carcinoma *in situ* (DCIS). *J. Theor. Biol.*, 301:122–40, 2012.
- A.J. Maniotis, C.S. Chen, and D.E. Ingber. Demonstration of mechanical connections between integrins cytoskeletal filaments, and nucleoplasm that stabilize nuclear structure. *P. Natl. Acad. Sci. USA*, 94(3):849–854, 1997.
- R.C. Martin, J.W. Newkirk, and R.S. Koss. *Agile Software Development: Principles, Patterns and Practices*. Pearson Prentice Hall, 2012.
- D. Marx and J. Hutter. *Ab Initio Molecular Dynamics: Basic Theory and Advanced Methods*. Cambridge University Press, 2012.
- MATLAB. *version 7.10.0 (R2010a)*. The MathWorks Inc., Natick, Massachusetts, 2010.
- F.A. Meineke, C.S. Potten, and M. Loeffler. Cell migration and organization in the intestinal crypt using a lattice-free model. *Cell Prolif.*, 34:253–266, 2001.
- F. Merchant, L.F. Bic, and M.B. Dillencourt. Load balancing in individual-based spatial applications. In *International Conference on Parallel Architectures and Compilations Techniques, Proceedings*, pages 350–357, 1998.
- N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *J. Chem. Phys.*, 21(6):1087–1092, 1953.
- G.R. Mirams, A.G. Fletcher, P.K. Maini, and H.M. Byrne. A theoretical investigation of the effect of proliferation and adhesion on monoclonal conversion in the colonic crypt. *J. Theor. Biol.*, 312:143–156, 2012.
- G.R. Mirams, C.J. Arthurs, M.O. Bernabeu, R. Bordas, J. Cooper, A. Corrias, Y. Davit, S.-J. Dunn, A.G. Fletcher, D.G. Harvey, M.E. Marsh, J.M. Osborne, P. Pathmanathan, J. Pitt-Francis, J. Southern, N. Zemzemi, and D.J. Gavaghan. Chaste: An open source C++ library for computational physiology and biology. *PLoS Comput. Biol.*, 9(3):e1002970, 2013.
- P.M. Morse. Diatomic molecules according to the wave mechanics. II. Vibrational levels. *Phys. Rev.*, 34:57–64, 1929.
- P.J. Murray, C.M. Edwards, M.J. Tindall, and P.K. Maini. From a discrete to a continuum model of cell dynamics in one dimension. *Phys. Rev. E*, 80(3), 2009.
- P.J. Murray, A. Walter, A.G. Fletcher, C.M. Edwards, M.J. Tindall, and P.K. Maini. Comparing a discrete and continuum model of the intestinal crypt. *Phys. Biol.*, 8(2):026011, 2011.
- P.J. Murray, C.M. Edwards, M.J. Tindall, and P.K. Maini. Classifying general nonlinear force laws in cell-based models via the continuum limit. *Phys. Rev. E*, 85(2, 1):021921, 2012.

-
- T. Nagai and H. Honda. A dynamic cell model for the formation of epithelial tissues. *Philos. Mag. B*, 81:699–719, 2001.
- I.S. Näthke. The adenomatous polyposis coli protein: The achilles heel of the gut epithelium. *Annu. Rev. Cell. Dev. Bi.*, 20(1):337–366, 2004.
- M.R. Nelson, D. Howard, O.E. Jensen, J.R. King, F.R.A.J. Rose, and S.L. Waters. Growth-induced buckling of an epithelial layer. *Biomech. Model. Mechanobiol.*, 10(6):883–900, 2011.
- T.J. Newman. Modeling multicellular systems using subcellular elements. *Math. Biosci. Eng.*, 2:613–624, 2005.
- G.M. Odell, G. Oster, P. Alberch, and B. Burnside. The mechanical bases of morphogenesis. I. Epithelial folding and invagination. *Dev. Biol.*, 85(2):446–462, 1981.
- J.M. Osborne, A. Walter, S.K. Kershaw, G.R. Mirams, A.G. Fletcher, P. Pathmanathan, D. Gavaghan, O.E. Jensen, P.K. Maini, and H.M. Byrne. A hybrid approach to multi-scale modelling of cancer. *Phil. Trans. R. Soc. A*, 368:5013–5028, 2010.
- M. Osterfield, X. Du, T. Schüpbach, E. Wieschaus, and S.Y. Shvartsman. Three-dimensional epithelial morphogenesis in the developing *Drosophila* egg. *Dev. Cell*, 24(4):400 – 410, 2013.
- E. Palsson and H.G. Othmer. A model for individual and collective cell movement in *Dictyostelium discoideum*. *Proc. Natl. Acad. Sci. USA*, 97(19):10448–10453, 2000.
- ParaView. ParaView web page, 2012. <http://www.kitware.com/products/paraview.html>.
- P. Pathmanathan, J. Cooper, A. Fletcher, G. Mirams, P. Murray, J. Osborne, J. Pitt-Francis, A. Walter, and S.J. Chapman. A computational study of discrete mechanical tissue models. *Phys. Biol.*, 6:036001+, 2009.
- J. Pitt-Francis and J. Whiteley. *Guide to Scientific Computing in C++*. Springer, 2012.
- J. Pitt-Francis, M.O. Bernabeu, J. Cooper, A. Garny, L. Momtahan, J. Osborne, P. Pathmanathan, B. Rodriguez, J.P. Whiteley, and D.J. Gavaghan. Chaste: using agile programming techniques to develop computational biology software. *Phil. Trans. R. Soc. A*, 366:3111–3136, 2008.
- S. Plimpton. Fast parallel algorithms for short-range molecular dynamics. *J. Comput. Phys.*, 117:1–19, 1995.
- S. Plimpton and B. Hendrickson. A new parallel method for molecular dynamics simulation of macromolecular systems. *J. Comput. Chem.*, 17:326–337, 1994.
- S.J. Plimpton and A.P. Thompson. Computational aspects of many-body potentials. *MRS Bull.*, 37:513–521, 2012.
- C.S. Potten, C. Booth, and D.M. Pritchard. The intestinal epithelial stem cell: the mucosal governor. *Int. J. Exp. Pathol.*, 78(4):219–243, 1997.

-
- S. Preston, S. McDonald, D. Oukrif, L. Greaves, D. Turnbull, P. Tadrous, P. Sasieni, J. Jankowski, M. Novelli, and N. Wright. Clusters of phenotypically related human colonic crypts develop through crypt fission: Implications for colorectal carcinogenesis. *Gastroenterology*, 128(4):A493, 2005.
- Python. Python language reference, Version 2.7, 2013. <http://www.python.org>.
- J.N. Reddy. *Introduction to the Finite Element Method*. McGraw-Hill Science/Engineering/Math, 2nd edition, 1993.
- P. Richmond, D. Walker, S. Coakley, and D. Romano. High performance cellular level agent-based simulation with FLAME for the GPU. *Brief. Bioinform.*, 11(3):334–347, 2010.
- C.I. Rodrigues, D.J. Hardy, J.E. Stone, K. Schulten, and W.M.W. Hwu. GPU acceleration of cutoff pair potentials for molecular modeling applications. In *Proceedings of the 5th Conference on Computing Frontiers*, pages 273–282, 2008.
- G. Rogers and N. Goel. Computer simulation of cellular movements: cell-sorting, cellular migration through a mass of cells and contact inhibition. *J. Theor. Biol.*, 71:141–166, 1978.
- T. Roose, S.J. Chapman, and P.K. Maini. Mathematical models of avascular tumor growth. *SIAM Rev.*, 49:179–208, 2007.
- R. Rucker and J. Walker. Cellab website. <http://www.fourmilab.ch/cellab/>.
- S.A. Sandersius and T.J. Newman. Modeling cell rheology with the Subcellular Element Model. *Phys. Biol.*, 5:015002, 2008.
- S.A. Sandersius, C.J. Weijer, and T.J. Newman. Emergent cell and tissue dynamics from subcellular modeling of active biomechanical processes. *Phys. Biol.*, 8:045007, 2011.
- T. Sato, R.G. Vries, H.J. Snippert, M. van de Wetering, N. Barker, D.E. Stange, J.H. van Es, A. Abo, P. Kujala, P.J. Peters, and H. Clevers. Single Lgr5 stem cells build crypt-villus structures *in vitro* without a mesenchymal niche. *Nature*, 459(7244):262–U147, 2009.
- H.M. Sauro, M. Hucka, A. Finney, C. Wellock, H. Bolouri, J. Doyle, and H. Kitano. Next generation simulation tools: The Systems Biology Workbench and BioSPICE integration. *OMICS*, 7:355–372, 2003.
- G. Schaller and M. Meyer-Hermann. Multicellular tumor spheroid in an off-lattice Voronoi-Delaunay cell model. *Phy. Rev. E*, 71:051910+, 2005.
- W. Schroeder, K. Martin, and B. Lorensen. *Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*. Kitware, 4th edition, 2006.
- K. Smallbone, R.A. Gatenby, R.J. Gillies, P.K. Maini, and D.J. Gavaghan. Metabolic changes during carcinogenesis: Potential impact on invasiveness. *J. Theor. Biol.*, 244(4):703–713, 2007.

-
- A.M. Smith, R.E. Baker, D. Kay, and P.K. Maini. Incorporating chemical signalling factors into cell-based models of growing epithelial tissues. *J. Math. Biol.*, 65(3):441–463, 2012.
- R.K. Standish and D. Madina. ClassdescMP: Easy MPI programming in C++. In Sloot, P.M.A. and Abramson, D. and Bogdanov, A.V. and Dongarra, J.J. and Zomaya, A.Y. and Gorbachev, Y.E., editor, *Computational Science, PT IV, Proceedings*, volume 2660 of *Lecture Notes In Computer Science*, pages 896–902, 2003. International Conference on Computational Science, Melbourne, Australia.
- M.S. Steinberg. The problem of adhesive selectivity in cellular interactions. *Cellular Membranes in Development. 22nd Symposium of the Society for the Study of Development and Growth*, 1964.
- J.E. Stone, J.C. Phillips, P.L. Freddolino, D.J. Hardy, L.G. Trabuco, and K. Schulten. Accelerating molecular modeling applications with graphics processors. *J. Comput. Chem.*, 28:2618–2640, 2007.
- B. Stroustrup. *C++ Programming Language*. Addison-Wesley Professional, 3rd edition, 1997.
- B. Stroustrup. Software Development for Infrastructure. *Computer*, 45(1):47–58, 2012.
- E. Süli and D.F. Mayers. *An Introduction to Numerical Analysis*. Cambridge University Press, 1st edition, 2003.
- D. Sulsky, S. Childress, and J. Percus. A model of cell sorting. *J. Theor. Biol.*, 106: 275–301, 1984.
- T. Sun, S. Adra, R. Smallwood, M. Holcombe, and S. MacNeil. Exploring hypotheses of the actions of TGF- β -1 in epidermal wound healing using a 3D computational multiscale model of the human epidermis. *PLoS ONE*, 4(12):e8515, 2009.
- M.H. Swat, G.L. Thomas, J.M. Belmonte, A. Shirinifard, D. Hmeljak, and J.A. Glazier. Chapter 13 - Multi-Scale Modeling of Tissues Using CompuCell3D. In A.R. Asthagiri and A.P. Arkin, editors, *Computational Methods in Cell Biology*, volume 110 of *Methods in Cell Biology*, pages 325 – 366. Academic Press, 2012.
- C.R. Sweet, S. Chatterjee, Z. Xu, K. Bisordi, E.D. Rosen, and M. Alber. Modelling platelet-blood flow interaction using the subcellular element Langevin method. *J. R. Soc. Interface*, 8(65):1760–1771, 2011.
- A.N. Tantawi and D. Towsley. Optimal static load balancing in distributed computer systems. *J. ACM*, 32(2):445–465, 1985.
- I.M.M. van Leeuwen, G.R. Mirams, A. Walter, A. Fletcher, P. Murray, J. Osborne, S. Varma, S.J. Young, J. Cooper, B. Doyle, J. Pitt-Francis, L. Momtahan, P. Pathmanathan, J.P. Whiteley, S.J. Chapman, D.J. Gavaghan, O.E. Jensen, J.R. King, P.K. Maini, S.L. Waters, and H.M. Byrne. An integrative computational model for intestinal tissue renewal. *Cell Prolif.*, 42:617–636, 2009.

-
- I.M.M. van Leeuwen, H.M. Byrne, O.E. Jensen, and J.R. King. Crypt dynamics and colorectal cancer: advances in mathematical modelling. *Cell Prolif.*, 39(3):157–181, 2006.
- L. Verlet. Computer “experiments” on classical fluids. I. thermodynamical properties of Lennard-Jones molecules. *Phys. Rev.*, 159:98–103, 1967.
- J. von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Champaign, IL, USA, 1966.
- D.C. Walker, J. Southgate, G. Hill, M. Holcombe, D.R. Hose, S.M. Wood, S. MacNeil, and R.H. Smallwood. The epitheliome: agent-based modelling of the social behaviour of cells. *Biosystems*, 76(1-3):89–100, 2004.
- D.C. Walker, S. Wood, J. Southgate, M. Holcombe, and R.H. Smallwood. An integrated agent-mathematical model of the effect of intercellular signalling via the epidermal growth factor receptor on cell proliferation. *J. Theor. Biol.*, 242(3):774–789, 2006.
- D.C. Walker, N.T. Georgopoulos, and J. Southgate. From pathway to population - a multiscale model of juxtacrine EGFR-MAPK signalling. *BMC Syst. Biol.*, 2(102), 2008.
- D.C. Walker, N.T. Georgopoulos, and J. Southgate. Anti-social cells: Predicting the influence of E-cadherin loss on the growth of epithelial cell populations. *J. Theor. Biol.*, 262(3):425–440, 2010.
- Y. Wang, Z. Khan, M. Kaschube, and E.F. Wieschaus. Differential positioning of adherens junctions is associated with initiation of epithelial folding. *Nature*, 484:390–393, 2012.
- W.M. Wong, N. Mandir, R.A. Goodlad, B.C.Y. Wong, S.B. Garcia, S.K. Lam, and N.A. Wright. Histogenesis of human colorectal adenomas and hyperplastic polyps: the role of cell proliferation and crypt fission. *Gut*, 50(2):212–217, 2002.
- F. Wottawah, S. Schinkinger, B. Lincoln, R. Ananthakrishnan, M. Romeyke, J. Guck, and J. Käs. Optical rheology of biological cells. *Phys. Rev. Lett.*, 94:098103, 2005.
- J. Xu, Y. Ren, W. Ge, X. Yu, X. Yang, and J. Li. Molecular dynamics simulation of macromolecules using graphics processing units. *Mol. Simulat.*, 36:1131–1140, 2010.
- B.B. Youssef, G. Cheng, K. Zygorakis, and P. Markenscoff. Parallel implementation of a cellular automaton modeling the growth of three-dimensional tissues. *Int. J. High Perform. Comput. Appl.*, 21:196–209, 2007.