An Algorithm and VLSI Architecture for a Stochastic Particle-Based Biological Simulator

Laurier Boulianne

Master Thesis

Department of Electrical and Computer Engineering

McGill University

Montreal, Quebec

2010-10-18

A thesis submitted to McGill University in partial fulfillment
of the requirements of the degree of Master of Engineering

ACKNOWLEDGEMENTS

Foremost, I would like to express my sincere gratitude to my supervisor, Prof. Warren J. Gross. Without his continuous support, knowledge and insight, this thesis would not have been possible. Over the few years spent working on this thesis, I also greatly appreciated his availability, his human approach and his expertise. I would like to thank Prof. Michel Dumontier who, together with Prof. Warren J. Gross, presented me the idea of developing a grid-based stochastic biological simulator. From that point on, Prof. Michel Dumontier provided me with constant assistance and guidance in my research, especially regarding any questions related to systems biology and bioinformatics. Finally, I am extremely grateful to my parents and family who supported me and provided me with encouragements during the hard days inevitably encountered during graduate research.

ABSTRACT

With the recent progress in both computer technology and systems biology, it is now possible to simulate and visualise biological systems virtually. It is expected that realistic in silico simulations will enhance our understanding of biological processes and will promote the development of effective therapeutic treatments. Realistic biochemical simulators aim to improve our understanding of biological processes that could not be, otherwise, properly understood in experimental studies. This situation calls for increasingly accurate simulators that take into account not only the stochastic nature of biological systems, but also the spatial heterogeneity and the effect of crowding of biological systems. This thesis presents a novel particle-based stochastic biological simulator named GridCell. It also presents a novel VLSI architecture accelerating GridCell between one and two orders of magnitude. GridCell is a three-dimensional simulation environment for investigating the behaviour of biochemical networks under a variety of spatial influences including crowding, recruitment and localisation. GridCell enables the tracking and characterisation of individual particles, leading to insights on the behaviour of low copy number molecules participating in signalling networks. The simulation space is divided into a discrete 3D grid that provides ideal support for particle collisions without distance calculations and particle searches. SBML support enables existing networks to be simulated and visualised. The user interface provides intuitive navigation that facilitates insights into species behaviour across spatial and temporal dimensions. Crowding effects on a Michaelis-Menten system are simulated and results show they can have a huge impact on the effective rate of product formation. Tracking millions of particles is extremely computationally expensive and in order to run whole cells at the molecular resolution in less than 24 hours, a commonly expressed goal in systems biology, accelerating GridCell with parallel hardware is required. An FPGA architecture combining pipelining, parallel processing units and streaming is presented. The architecture is scalable to multiple FPGAs and the streaming approach ensures that the architecture scales well to very large systems. An architecture containing 25 processing units on each stage of the pipeline is synthesised on a single Virtex-6 XC6VLX760 FPGA device and a speedup of 76x over the serial implementation is achieved. This speedup reduces the gap between the complexity of cell simulation and the processing power of advanced simulators. Future work on GridCell could include support for highly complex compartment and high definition particles.

ABRÉGÉ

Grâce aux récents progrès en informatique et en biologie, il est maintenant possible de simuler et de visualiser des systèmes biologiques de façon virtuelle. Il est attendu que des simulations réalistes produites par ordinateur, in silico, nous permettront d'améliorer notre connaissance des processus biologiques et de favoriser le développement de traitements thérapeutiques efficaces. Les simulateurs biologiques visent à améliorer notre connaissance de processus biologiques qui, autrement, ne pourraient pas être correctement analysés par des études expérimentales. Cette situation requiert le développement de simulateurs de plus en plus précis qui tiennent compte non seulement de la nature stochastique des systèmes biologiques, mais aussi de l'hétérogénéité spatiale ainsi que des effets causés par la grande densité de particules présentes dans ces systèmes. Ce mémoire présente GridCell, un simulateur biologique stochastique original basé sur une représentation microscopique des particules. Ce mémoire présente aussi une architecture parallèle originale accélérant GridCell par presque deux ordres de magnitude. GridCell est un environnement de simulation tridimensionnel qui permet d'étudier le comportement des réseaux biochimique sous différentes influences spatiales, notamment l'encombrement moléculaire ainsi que les effets de recrutement et de localisation des particules. GridCell traque les particules individuellement, ce qui permet d'explorer le comportement de molécules participants en très petits nombres à divers réseaux de signalisation. L'espace de simulation est divisé en une grille 3D discrète qui permet de générer des collisions entre les particules sans avoir à faire de calculs de distance ni de recherches de particules complexes. La compatibilité avec le format SBML permet à des réseaux déjà existants d'être simulés et visualisés. L'interface visuelle permet à l'utilisateur de naviguer de façon intuitive dans la simulation afin d'observer le comportement des espèces à travers le temps et l'espace. Des effets d'encombrement moléculaire sur un système enzymatique de type Michaelis-Menten sont simulés, et les résultats montrent un effet important sur le taux de formation du produit. Tenir compte de millions de particules à la fois est extrêmement demandant pour un ordinateur et, pour pouvoir simuler des cellules complètes avec une résolution spatiale moléculaire en moins d'une journée, un but souvent exprimé en biologie des systèmes, il est essentiel d'accélérer GridCell à l'aide de matériel informatique fonctionnant en parallèle. On propose une architecture sur FPGA combinant le traitement en pipeline, le fonctionnement en mode continu ainsi que l'exécution parallèle. L'architecture peut supporter plusieurs FPGA et l'approche en mode continu permet à l'architecture de supporter très grands systèmes. Une architecture comprenant 25 unités de traitement sur chaque étage du pipeline est synthétisée sur un seul FPGA Virtex-6 XC6VLX760, ce qui permet d'obtenir des gains de performance 76 fois supérieurs à l'implémentation séquentielle de l'algorithme. Ce gain de performance réduit l'écart entre la complexité de la simulation des cellules biologiques et la puissance de calcul des simulateurs avancés. Des travaux futurs sur GridCell pourraient avoir pour objectif de supporter des compartiments de forme très complexe ainsi que des particules haute définition.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
Introduction

1.1 Systems Biology

Computational cell biology is currently one of the most exciting cross-disciplinary areas of research [31]. An ambitious, long term goal of this field is to produce accurate simulations of biological cells with molecular resolution. This task has two major challenges: 1) the construction of accurate biological models and 2) the development of scalable simulation architectures. Efforts towards realising these objectives are expected to support therapeutic drug development against human diseases by increasing our understanding of biological systems.

In order to construct a model that is as accurate as possible, the simulator has to be compatible with several key aspects of molecular systems. First, biological systems are inherently stochastic and spatially dependent. Second, it is well known that molecules behave with Brownian dynamics [27]. Also, the functionality of certain proteins known as enzymes is limited by the rate of diffusion in the solution medium. An important aspect to this functionality has to do with the mobility of particles in what is increasingly believed to be a crowded environment [35]. Importantly, certain cellular responses occur as a result of single or few particle fluctuations, and this precludes the modelling of systems with continuum dynamics. Finally, the effect of spatial localisation is expected to play an important role in the behaviour of the system [31]. The idealisation of a "well-mixed" system is unlikely to reflect biological reality, where molecular complexes form scaffolds of recruitment for cellular signalling and metabolism. Indeed, stochastic and spatial considerations are necessary for realistic in silico simulation of biological cells.

Another key aspect that is required to build an accurate model is to possess enough biological knowledge to be able to describe with enough details the behaviour of the system. Such information is rarely totally known and often slow and difficult to acquire. As such, the general approach relies on formulating hypotheses and validating results in silico with known experimental facts. By using the knowledge acquired from simpler and smaller system, it is possible to build larger and more complex simulations to achieve a better understanding of the biological processes in a living cell.

Simulating a biological cell at the molecular level implies dealing with millions of particles moving, reacting and colliding with each other, often simultaneously. It also implies having to handle thousands of different reactions in different compartments. Consequently, the simulator has to be efficient and scalable. One key aspect that can be used to improve the scalability of the system is to exploit the fact that while particles close to each other are dependent, they only influence particles in their neighbourhood and, therefore, particles that are far away are independent and can be processed concurrently. An extremely computationally expensive problem with a large amount of independent data is a situation which applies to several other fields, such as video graphics processing, traffic simulations [53], particle physics, molecular dynamics [4][11], and more. The traditional solution to these problems is to implement those algorithms on massively parallel platforms such as graphic processing units (GPUs), field programmable gate array (FPGA) devices, supercomputers and computer clusters.

1.2 Motivation

The simulation of discrete, stochastic, spatially-dependent molecular systems is extremely computationally expensive and current simulators can not support all of these characteristics without being too slow to be usable on large systems. In addition, molecular crowding, which is believed to have a large impact in biological networks, requires a microscopic representation of the particles as well as collision detection between particles. This feature is rarely supported by the recent simulators due to its high computational cost, and most simulators have opted for a dimensionless representation of the particles.

Hardware acceleration might close the gap between the high computational cost of cell simulation at molecular resolution and the processing power of current computers. Many algorithms which can benefit from parallel computing have gained speedups between one and two orders of magnitude when ported to FPGA architectures. While some simulators are planning to accelerate their algorithm over multiple computers, the massively parallel approach has not yet been fully explored. Moreover, no stochastic and spatial simulators have yet been designed with parallel computing in mind, nor been implemented in a massively parallel environment.

This is why we present GridCell, a stochastic, spatial, particle-based simulator with a highly scalable architecture.

1.3 Proposed Research

We developed GridCell to simulate biological models with specific considerations for stochasticity, locality, and collision [7][6]. GridCell is based on a simplified model for molecular movement and interaction. The organisation of the space is heavily based on the cellular automata model which is inherently heavily parallelisable. It uses a discrete three-dimensional cubic grid based on

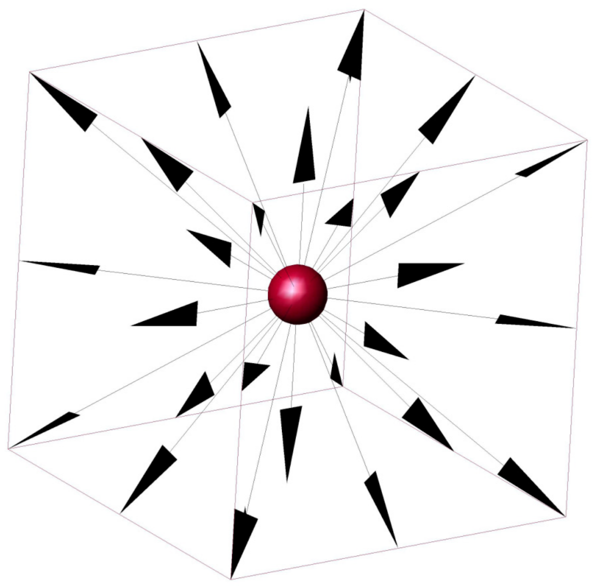

Figure 1–1: The 27 different directions of the D3Q27 grid.

the D3Q27 model often used in the application of the Lattice-Boltzmann Method (LBM) [59]. Each discrete location of the grid, called a voxel, has access to itself and its 26 neighbours and is independent of the other voxels outside this immediate neighbourhood. Figure 1–1 shows the 27 different locations accessible to a voxel from a D3Q27 grid. The integer-addressed 3D grid avoids floating-point computations and distance calculations, resulting in an efficient implementation. Molecules are represented as particles that move and react stochastically within discrete volumes in discrete time steps. Collisions and molecular crowding are enforced since only one particle can occupy a given location at any time. GridCell stores the coordinates of all the particles on the 3D grid at every turn, thereby enabling individual particle tracking in both space and time. An OpenGL user interface shows the particles in the 3D space and plots concentration and surface graphs. GridCell's biological models are written and stored by following the standard Systems Biology Markup Language (SBML) already used by several hundreds of other applications. Spatial information of the system which is not inherently supported by the SBML format is entered into GridCell with specific custom annotations. The state of the systems can also be saved in a tab delimited text file to be used by other tools for analysis and imaging. The algorithm provides biologically valid results for well-mixed systems and provides additional insight in crowded and spatially heterogeneous systems.

GridCell performance is tightly linked to the number of voxels in the simulation space. The software simulator can currently support a maximum of 10^7 voxels/particles which is not enough to simulate structures as complex as a complete cell at a molecular resolution, the long-term goal of GridCell. However, the simple and regular algorithm of GridCell, which does not require any searches or complex operations, is a prime candidate for acceleration by parallelisation. An FPGA architecture was developed and is presented in Chapter 4. The architecture benefits from the regularity, simplicity and independence of non-adjacent particles to process several particles at the same time using multiple processing elements. The throughput of the pipelined architecture is larger by almost two orders of magnitude compared to the serial implementation of the algorithm. This large acceleration closes the gap even further between full cell simulations at the molecular resolution and current available processing power.

1.4 Outline

This thesis is divided in several chapters. Chapter 2 provides an overview of the research done over the past 40 years in systems biology. From the early qualitative models to the deterministic solvers, to the stochastic simulation algorithm (SSA) and, finally, the latest spatial simulators. The origins and uses of the highly parallel cellular automata model, which GridCell is based upon, is also reviewed. The popular parallel platforms that can be used to accelerate GridCell are also described, as well as a brief overview of the literature on some applications previously accelerated. The details of the GridCell's algorithm are described in Chapter 3, along with a description of its user interface, the format of the input biological models and results from biological simulations under various conditions. Chapter 4 describes the details of the 6-stage pipeline of the FPGA implementation of the algorithm. The results from the parallel architecture are provided, showing large increases in performance. Chapter 5 mentions the future work that can be performed on either the software, the algorithm itself or the very-large-scale integration (VLSI) architecture. Chapter 6 concludes the thesis by reviewing the current situation of systems biology along with what can be expected in the near future.

1.5 Previous Papers and Contributions

Related contributions:

1. L. Boulianne, S. Al Assaad, M. Dumontier, W. J. Gross. GridCell:A stochastic particle-based biological system simulator, *BMC Systems Biology*, 2(66), July 2008.
2. L. Boulianne, M. Dumontier, W. J. Gross. A Stochastic Particle-Based Biological System Simulator, in *Proc. 2007 Summer Computer Simulation Conference*, San Diego, California (USA), July 2007, pp.794-801.

3. L. Boulianne, M. Dumontier, W. J. Gross. GridCell :A Stochastic Particle-Based Simulator, poster presentation, *Ottawa Institute of Systems Biology (OISB)*, Ottawa, Ontario (Canada), November 2006.

Non-related contributions:

1. L. Boulianne, W. J. Gross. SIMD implementation of interpolation in algebraic soft-decision Reed-Solomon decoding, *IEEE Workshop on Signal Processing Systems Design and Implementation (SIPS 2005)*, Athens, Greece, November 2005, pp. 750-755.
2. L. Boulianne, W. J. Gross. DSP Implementation of a Soft-Decision Reed-Solomon Decoder, poster presentation, *Micronet Anuual Workshop 2005*, Ottawa, Ontario (Canada), 2005.

CHAPTER 2
Background

Biology is the natural science studying life, which is the study of self-replicating processes. Living organisms have the ability to grow, reproduce, adapt and evolve. Biology is one of the most diverse fields of study and is divided into many subbranches which all focus on a specific aspect of life, including anatomy, biochemistry, ecology, genetics, neurobiology, cell biology, bioinformatics, systems biology and many more. Biology is mainly an experimental science as the evolutionary processes of living organisms are not yet understood well enough to allow for theoretical advances.

The recently acquired knowledge on how life grows, sustains and replicates created a large impact on the life of mankind. Advances in alimentation, drug development, hygiene, medicine and surgical procedures have made it possible to live longer, healthier and created new possibilities of recovering from and preventing injuries, diseases and malformations. The health sector, which is mostly derived from the knowledge acquired in biology, is one of the biggest industry of developed countries. For example, Canada spends more than 10% of its total Growth Domestic Product (GDP) on Health Care, which corresponds to more than 30G$ per year, while the pharmaceutical and medicine manufacturing industry alone is generating sales larger than 5G$ [46]. Even with all the investments and energy spent in the general field of biology, the overall knowledge of the living processes is fairly limited and a lot still needs to be understood, discovered and recorded. The recent technological advances of computers and measurement and imaging tools allow the development of new models which can provide a better understanding of the processes inherent to life. These new tools and models are believed to support therapeutic drug development.

2.1 Qualitative Models

Prior to the recent technological advances in the field of computing, the use of qualitative models to describe biological processes was common (and it still is). Some examples of qualitative models include graphs, laws, diagrams, relationships, Bayesian inference, stoichiometric and constraint-based modelling and several others. The main advantages of these qualitative models are that they usually do not require extensive calculations nor extensive biological background in order to provide additional insights on a particular biological network.

2.2 Systems Biology

Systems biology is the quantitative study of biological systems [50]. It is a new branch of biology that has been made possible by the recent technological advances and with the arrival of extremely large amount of biological data gathered with new tools over the last 15 years. Systems biology is in a state called data-rich but hypothesis-poor, which means that a lot of biological data is available, but not much understanding of biological processes is made out of it. Even in this state, most data required to completely describe the elements of a given system is still unknown or not yet recorded. This is why a purely reductionist approach has not been successful. In order to get a better knowledge of biological systems, neither a purely deduction nor a purely induction approach can be used. Instead, an iterative cycle, named the cycle of knowledge in [50] is more appropriate. This circle iterates over two different stages. The first one builds from limited knowledge and ideas to create an hypothesis, then performs an analysis to generate new data and observations. These newly acquired data and observations are then synthesised through an induction process into new knowledge and ideas. The cycle can then start over with this newly acquired knowledge. The use of mathematical analysis and computer simulations can demonstrate that a given hypothesis is either valid or invalid and can provide quantitative data used for future experimental exploration. As such, they are an important element of the iterative process of the circle of knowledge.

The new processing power and knowledge of cell mechanics formed a new branch in systems biology called computational cell biology. Computational cell biology explores the dynamic processes of the living cells. One key characteristic of these processes is that even the simplest ones are extremely complex and are inherently non-linear, which make them difficult to solve analytically. Instead, numerical solutions, solved by mathematical models on computers have proven to be far more efficient. These simulators have been an important tool in understanding biological mechanics and in refining the kinetic model taking part inside living cells [13]. Virtual simulators can explore hypothesises that are otherwise difficult or impossible to analyse in laboratory experimentation, and they are also much cheaper to maintain and execute than actual laboratory experiments. Finally, a commonly expressed goal in computational cell biology is to be able to run one-day simulation of a whole cell at the molecular resolution [31].

2.3 General-Purpose Simulators

While many models were designed to target specific applications, such as DNA pattern matching, protein folding molecular dynamics (MD), neuronal interconnections, ca^{2+} ions dispersion simulations and more, we are interested in the category of simulators described as general purpose simulators, which explores the middle-ground scale of the cell. General purpose simulators aim to include as many details as possible to accurately describe the interactions between species inside a cell while keeping the level of details as low as possible to keep the complexity at a reasonable level. For example, modelling the molecular dynamics of particles in order to determine if a reaction is successful is considered to take the bottom-end approach of exploring the behaviour of biomolecules. While it is very accurate, the scale in size and time of current simulations is too small to encompass the whole cell. Therefore, including this level of details is usually out of the scope of general purpose simulators. Instead, most middle-ground simulators use the less complex and more general rate of reaction coming from the law of mass action, which corresponds to the averaged behaviour resulting from collisions between particles. On the other hand, modern simulators includes characteristics such as stochastic events, tridimensional space and compartments as well as individual particle diffusion movements, which are not usually included in the top-end simulation approaches exploring the system-level behaviour of biological processes. General purpose simulators are probably the best suited class of simulators able to capture the behaviour of a whole cell at molecular resolution.

The first general purpose simulators were fairly simple and many assumptions had to be made, such as well-mixed assumptions, continuous concentrations, dimensionless compartments, etc. As time passed and computer power and biological knowledge increased, characteristics such as stochasticity, 3D space, diffusion, compartment geometry started to be included in recent simulators. We will look at the popular general purpose biological simulators along with their principal characteristics.

2.3.1 Ordinary Differential Equation Solvers

Chemical reactions have traditionally been simulated by solving a set of ordinary differential equations (ODEs) as they are the inherent solution of well-mixed kinetics models [13]. Let's take a simple model representing a protein, which can be in either a closed or an open state. The kinetics of such a model is usually described as

$$C \Leftrightarrow O, \tag{2.1}$$

where C represents the closed state and O the open state of the protein. The bidirectional arrow means the reaction (or the change of state) is reversible. A closed protein is assumed to change to its open state at a given rate J^+. Similarly the open protein closes at a rate J^-. It is assumed that a large number of proteins are present in the system and the rates represent the average number of transformations per unit of time. The rates are determined by the value of the so-called "law of action mass", which comes from the early days of the studies of chemical kinetics and stipulates that the number of reactions is proportional to the product of the concentration of the reactants involved in the reaction multiplied by the rate constant. By defining the forward rate constant as k^+ and the reverse rate constant as k^-, the corresponding overall rates are given by $J^+ = k^+[C]$ and $J^- = k^-[O]$, where the brackets [] represent the concentration of the protein in the corresponding state. By applying the law of conservation of mass, we can affirm that the total number of proteins N is the sum of the closed proteins N_c and the open proteins N_o. Since N_c can be expressed as $N_c = N - N_o$, solving the problem for N_o is also solving the problem for N_c. In biological systems, the concentration can be measured in several different ways, such as in terms of cell volumes, weights or raw numbers of molecules. Here, we define the concentration as the number of particles in the given state over the total number of particles N so that $[O] = f_o = N_o/N$. We can also define the proportion of closed proteins $f_c = 1 - f_o$. Multiplying both concentrations f_o and f_c with their respective rate constant k^- and k^+ provides the flux for each process, which represents the change that each process makes for each state. We define the two fluxes with the symbols j^- and j^+ so that

$$flux(O \rightarrow C) = j^- = k^- f_o \tag{2.2}$$

$$flux(C \rightarrow 0) = j^+ = k^+(1 - f_o). \tag{2.3}$$

Both fluxes are linked together, and when one increases, the other decreases. The difference of the two fluxes represents the change of f_o over time.

$$\frac{df_o}{dt} = j^+ - j^- = k^+(1 - f_o) - k^- f_o = -(k^- + k^+)(f_o - \frac{k^+}{k^- + k^+}) \tag{2.4}$$

by substituing $\tau = \frac{1}{k^- + k^+}$ and $f_\infty = \frac{k^+}{k^- + k^+}$, we get

$$\frac{df_o}{dt} = \frac{-(f_o - f_\infty)}{\tau}, \tag{2.5}$$

which is a classical differential equation of the type

$$\frac{dX}{dt} = \frac{-X}{\tau}. \tag{2.6}$$

The solution to this kind of differential equation is well known and is of the type

$$X = Ce^{-\tau t}, \tag{2.7}$$

where C is a constant. From there, finalising the equation by using the initial conditions of the species solves the system for the concentration of open and closed proteins over time. This simple model can be solved analytically. However, even though it is possible to describe complex models as a set of differential equations, the solution is rarely obtainable analytically and numeric methods are needed. Thus, ODE solvers have been among the first biological simulators. They are fast and simple to use, and many systems have been successfully simulated with ODE solvers, such as the Michealis-Menten kinetics, systems with feedback loops (positive and negative), oscillatory networks and more. However, a lot of assumptions are made in order to create those differential equations. First, it is assumed that a well-mixed system where particles are uniformly distributed in a dimensionless compartment is used. The differential equations also assume there are enough particles so that the changes in concentration are continuous. Also, if the system is stable, differential equations always converge toward the same deterministic solution. All those assumptions are not reflecting the spatial organisation and chaotic behaviour of the particles within cellular compartments.

Partial differential equation (PDE) solvers can be used to include the concept of gradient concentration in the system. The state of the system is given as a density function with respect to space, and as long as the scale of the space and the number of particles involved is large enough so that the discrete properties of the individual particles can be neglected, they can provide informative results. Reaction-Diffusion systems can be described with PDEs and localisation effects can be observed at the cost of increased complexity. They are still used today by some simulators since they can be fast, robust and are relatively easy to set up.

2.3.2 Stochastic Simulation Algorithm

As mentioned above, continuum representation of the system is appropriate as long as the densities are large enough. In biological systems, the small populations of some reactant species can produce randomness and discrete effects in the behaviour of the living cells that cannot be reproduced by traditional ODE and PDE simulators. The importance of this noise in cellular pathways has usually been underestimated and it is only recently that its importance became apparent [38]. In order to explore this chaotic behaviour inside cells, a stochastic representation of the system is required. The stochastic simulation algorithm from Gillespie [18] is the first stochastic simulator created. The SSA uses a Monte Carlo (MC) strategy to solve the Chemical Master Equation (CME) describing the state of a biological system. It is still a dimensionless algorithm that assumes a well-mixed solution, as only the concentration of the species are tracked and not their location, but, unlike to ODE simulators, the SSA can produce stochastic effects. The algorithm considers a system with N different species interacting with M different reactions. $x = X(t) = (X_1(t), .., X_N(t))$ represents the vector containing the number of molecules of each species at time t given that the vector was initialised to some non-zero value at time $t = t_0$. The array v_{ij} is defined as the stoichiometry matrix where each vector v_j represents the change in population of species i when reaction j occurs so that when reaction j fires, the state of the system jumps to $x + v_j$. $a_j(x)$ is defined as the propensity function containing the probability that reaction j occurs within the next infinitesimal time dt given the state of the system x. Also, $a_0(x)$ is defined as being the sum of all $a_j(x)$. For unimolecular reactions involving species i, $a_j(x)$ equals $c_j x_i$ where c_j has the same numerical value as k_j, the rate constant used in the conventional continuous case. For bimolecular reactions involving species i and i_2, $a_j(x)$ is defined as $c_j x_i x_{i_2}$ where $c_j = k_j/V$ and V is the volume of the simulation space. The exact derivation of the update process from the CME is described in [50]. The idea is that instead of trying to solve the full probability densities described by the CME over time, which is impossible but for the simplest systems, the SSA follows a numerical realisation of the CME. This numerical realisation is equivalent to taking a single possible path out of the infinite number of states that are possible. By running the SSA multiple times and saving all the results in a histogram, the probability density function can be approximated. The selection of the reaction occurring at each time step τ is determined randomly, as well as the size of τ. The update process requires two uniform random numbers between zero and one, r_1 and r_2. The time step τ is inversely proportional to the value of $a_0(x)$ and is modified by a random factor driven by r_1, so that

$$\tau = \frac{1}{a_0(x)} ln(\frac{1}{r_1}). \tag{2.8}$$

The reaction j occurring during the time step τ is chosen by generating a uniform value between 0 and a_0. The first reaction j where the sum of the propensity values from $j_1 = 0$ up to j is larger than the random number is chosen. The exact equation is

$$j = smallest\ integer\ satisfying \sum_{j_1=1}^{j} a_{j_1}(x) > r_2 a_0(x). \tag{2.9}$$

The update of the SSA can be divided into 5 different steps:

- 1. Initialise t to t_0 and x to x_0.
- 2. Compute all the $a_j(x)$ as well as the sum $a_0(x)$.

- 3. Compute the next τ and reaction j according to equations 2.8 and 2.9.
- 4. Update the system so that $t = t + \tau$ and $x = x + v_j$.
- 5. Record the state of the system if needed and go back to step 2 if the simulation is not finished.

This implementation of the algorithm is the original algorithm and is called the Direct Method (DM). The major issue of the DM is that it is often very slow because the number of iterations scales linearly with the number of reactions and the time step τ is inversely proportional to $a_0(x)$, which is proportional to the size of the system. The Tau-Leaping method is an evolution of the DM that advances through time by a preselected amount of time τ and processes several reactions per time steps. The number of reactions taking place during this amount of time is estimated by a Poisson random variable. The Logarithmic Direct Method (LDM) [33] is another version of the SSA. These two adaptations of the SSA reduce the scaling issues of the original algorithm. The SSA has been used to simulate many different systems and is still popular for simulation assuming a well-mixed environment.

2.3.3 Characteristics of Modern Simulators

We discussed in the previous section that noise had an important effect on the signalling pathways of living cell and why a stochastic representation of the system is optimal. An aspect which is neglected with the SSA is the effect of protein localisation and mobility within the cell. The SSA assumes a well-mixed solution with dimensionless compartments which makes abstraction of molecular mobility and structural/compartmental effects. Macromolecular mobility in living cells is affected by what is called the molecular crowding. Molecular crowding is believed to be important in cellular organisms [16] and induces non-linear signal delays by causing anomalous diffusion speeds of macromolecules. Anomalous diffusion speed is defined as being a sub-linear scaling of mean-square displacement of the molecule over time [51]. This anomalous diffusion can have a huge impact in systems where the reaction rates are faster than the diffusion speeds and can modify the classical reaction kinetics. Limited mobility kinetics have been described as fractal kinetics [45]. The macromolecular density of a typical cell is also much higher than typical in vitro conditions, so even with laboratory experiments, it is hard to quantify and predict its effect. The molecular density inside a living cell is also variable, which suggests that an explicit representation of crowding effects would be superior to an implicit representation (which requires to manually reduce the diffusion speeds and constant rates or artificially add physical constructs to hinder the progress of particles). Beyond the mobility issues of macromolecules in cells, the idealisation of a well-mixed dimensionless system is unlikely to reflect biological reality where molecular complexes form scaffolds of recruitment for cellular signalling. The physical locations of the molecules have to be tracked, and tridimensional compartments should have an effect on the mobility and location of the macromolecules in order to simulate molecular crowding and localisation effects.

In order to overcome some of the limitations of the SSA, namely, the well-mixed assumption and dimensionless compartments, several new biological simulators have been developed over the past decade. These simulators adopted different strategies to describe the biological models, each having different pros and cons regarding accuracy, amount of details, ease of use and computational complexity. We will look at popular strategies used regarding model geometry, and how movements and reactions between species can be handled.

The integration of spatial effects can be implemented in many different ways. The simplest in term of computational power is called the compartmental model where a set of non-spatially resolved compartments are connected together. Flux of species can be set so that particles go in and out of the various compartments. While it is possible to observe some compartmental effects, it is impossible to track down the location of individual or group of particles as the compartments have no defined shape. Therefore, diffusion or molecular crowding can't be simulated.

Space can also be represented as a continuous volume limited by external walls where individual particles or densities of molecules occupy a floating point location. Individual particles can move following a Brownian random walk and react with nearby particles, while densities generally use a deterministic diffusion scheme. Compartments can have theoretically any shapes, from realistic models derived from experimental images to idealised models built with constructive solid geometry (CSG), to analytical shapes described by mathematical expressions. The compartments can also be constructed with meshes. This representation possesses a lot of flexibility and can represent the space geometry within living cells with high accuracy. However, the algorithm usually needs floating point calculations, while intermolecular reactions have to go through a search or collision detection algorithm to find nearby particles. This can increase the simulation computational complexity. Exclusion volume calculation, together with collision detection, can be performed to simulate crowding effects; it is however very expensive to do so and simulations with this amount of details have been limited to small systems. Most simulators have opted for a dimensionless representation of particles, which ignore molecular crowding effects.

The last type of space geometry is the lattice or discretised space. It is a quantisation of the continuous volume space where a finite number of distinct areas (in 2D) or volumes (in 3D) form a grid containing all the possible physical locations. With a fine enough grid, the quantisation effects can be ignored. The shape of the lattice can be almost anything from triangle and hexagons, to squares, cubes and more. The shape of the lattice can have an impact on the simulation results [54]. Structural compartments can be a discretised version of the continuous compartment and the same techniques can be applied to construct them. Using an integer-based lattice space reduces the need for floating point operations. Lattice space is also naturally compatible with cellular automata (CA) algorithms which are described in section 2.5.1. CA have several advantages, such as using a simple logic, being versatile and scalable, eliminating complex distance calculations by limiting interactions between neighbouring sites, being naturally parallel algorithms and providing simple collision detection schemes. The size of the lattice sites can be large enough to contain many particles; this technique permits faster execution time and smaller memory requirements at the cost of lower precision.

The species can be represented in three different ways. The first one is the microscopic representation, where each molecule is represented as an individual particle occupying a specific location. This is the most computationally expensive species description, but it is the one which provides the most details, as it is possible to observe, follow and define the specific behaviour of each individual particle. This representation is needed to explicitly simulate crowding effects. The mesocopic description represents species with populations of particles that are located at a specific location inside a discrete space. This allows quicker simulation execution time at the cost of losing the ability to track individual particles. Finally, the last one is the macroscopic species representation of particles, which describe the species as densities in a continuous space. This type of representation is often used by PDE simulators. It works well with large populations but, as noted previously, fails when dealing with smaller amounts of particles.

Unlike dimensionless algorithms, which only consider the concentration of the species, most spatial simulators add the notion of locality or “neighbourhoodness” between two species before triggering a reaction. In a continuous space with a microscopic representation of the particles, the hard-sphere model is often used [49][37]. This model assumes that each particle occupies a given spherical volume and that if two different particles are in touch with each other, the particles have collided and a reaction can occur. The rate of successful reactions is derived from the macroscopic rate of reaction. Similarly, in discretised space with a mesoscopic representation of the reacting species, a collision is assumed when two particles occupy the same location. If a microscopic particle representation is adopted, reaction occurs with neighbouring locations or membranes.

It is well known that molecules behave with Brownian dynamics [27]. This Brownian random walk leads to a diffusing behaviour of the particles that is based on the statistical law of Einstein-Smoluchowski (more details are provided in Section 3.1.1). In a microscopic continuous space, it is possible to emulate the Einstein-Smoluchowski equation with Gaussian random variables [37]. In a discrete space, particles jump to random nearby locations at a frequency determined by the diffusion speed of the particles.

2.3.4 Modern Simulators

Once the geometry, species description, movement and reaction procedures are determined, the algorithm must use the kinetic theory, the CME for stochastic algorithms and law of mass action for deterministic models and apply diffusion (fick's law) to translate the simulator into an actual valid algorithm which has some biological relevance. Of course, depending on the approximations and assumptions being made and the limitation in computation power (and appropriate data), some simulators perform better for some specific models and applications than others. Following is a quick overview of some of the most well-known and popular general purpose simulators along with their principal characteristics. Table 2–1 describes the primary characteristics of some of the most well-known and popular spatial simulators.

The Virtual Cell (VCell) [44] is a popular deterministic simulator solving PDEs with the finite volume method to compute the concentration and location of every species. The simulation space is divided into compartments that represent the cell spatial structure. The compartments can be subdivided into smaller subvolumes to increase accuracy at the cost of larger computational time. The software has a powerful user interface where the user can specify the topology of the cellular structure, the geometric model, the simulation results and much more. The software uses its own Virtual Cell Markup Language (VCML) to describe the models and has some compatibility with SBML. VCell is used to describe a multitude of models with high accuracy such has the calcium transport mechanisms of PC12 cells [12] or the spatial and temporal dynamics of chemotactic networks [60]. It is one of the most scalable algorithms; it is however a deterministic algorithm that can not simulate either stochastic or molecular crowding effects.

StochSim [30] has been developed as part of a study on bacterial chemotaxis as a way to explore the stochastic effects of this signalling pathway. It has been later modified to become a general purpose simulator. It is a stochastic simulator that tracks individual molecules on a discretised 2D grid. Species can be in different states such as phosphorylation, methylation, or other covalent modifications which can affect the reaction rates of the molecules. Simple 2D structures can be created where nearest-neighbour interactions of molecules can be simulated. Work is currently done to port the algorithm to a 3D grid.

MCell [49] is a stochastic simulator tracking individual particles in a continuous 3D space. The diffusing particles move independently with Brownian dynamics. 2D meshes form surfaces representing membranes, compartments and sites of chemical reactions which can handle highly detailed structures. The 2D structures are mapped in the 3D volume and a ray-tracing algorithm is used to detect collisions between particles and the surface, resulting in chemical reactions. Reactions occur only on surfaces. The models are described using their simple model description language (MDL) and the output can be visualised with the DReAMM tool (Design, Render and Animate MCell Models). Biomedical applications such has calcium dynamics [14] and ectopic neurotransmission [19] were modelled. Two means of parallelising the algorithm are currently being pursued, one with supercomputers and the other with the use of computer clusters.

SmartCell [2] and MesoRD [21] use a similar approach where they use a mesoscopic representation of the particles, which subdivides the simulation space into smaller discrete subvolumes (voxels) containing many particles. Each of these subvolumes is considered to be a well-mixed solution and have to be small enough to be considered homogenised by diffusion within the time-scale of the reactions. Inside each subvolume, an SSA-like procedure is performed to update the system. The particles can diffuse to adjacent subvolumes. Simulation of the Min-System in E.Coly has been performed in MesoRD. This approach allows quicker simulations, but it is impossible to track individual particles, and molecular crowding has no effect on movement and reaction rates.

Cell++ [43] uses two different particle representations. Small particles (which are in large numbers) are simulated with a mesoscopic representation using a cellular automata engine with Brownian dynamics on a discretised grid. The larger molecules are simulated, microscopically and stochastically, on a continuous space. Both spaces are then superimposed onto each other, and reactions can take place between the two different spaces. This hybrid solution combines the fast execution of large quantities of diffusive particle from the mesoscopic representation with the details included with the microscopic representation of the larger particles. An openGL interface provides a 3D internal view of the simulation space. Case studies such has signal transduction pathway (MAPK cascade), methabolic pathways (glycolysis) and intracellular calcium signalling effects were simulated. All molecules are considered as pointless particles and only collisions between particles and fixed membranes separating two compartments are supported.

Table 2–1: Spatial Simulators.

	GridCell	SmartCell	MesoRD	Cell++	MCell	Smoldyn	ChemCell
Molecule representation	Particle	Population	Population	Hybrid	Particle	Particle	Particle
Stochastic	Yes	Yes	Yes	Large particles only	Yes	Yes	Yes
Space	Discretised	Discretised	Discretised	Continuous and Discretised	Continuous	Continuous	Continuous
Explicit Molecular Crowding	Yes	No	No	No	No	No	No
Diffusion support	Yes	Yes	Yes	Yes	Yes	Yes	Yes
SBML support	Yes	Yes	Yes	No	No	No	No
Web availability	Yes	Yes	Yes	Yes	Yes	Yes	Yes

ChemCell [37] is a particle-based (microscopic) simulator where each particle is individually located on a continuous space. The particles diffuse in the 3D spaces by following a Brownian motion and are represented as dimensionless particles. Nearby particles can react with each other and the probability of reaction per time step is calculated from the macroscopic reaction rate. The geometry of the cell is done with constructive solid geometry or as an assemblage of triangles meshed together. The creation of the cellular geometry and the output visualisation are handled by external tools. An ODE or SSA solver can also be used when spatial effects are not wanted. The simulation space can be partitioned among multiple computers to allow for parallel processing. Various effects such as chemotaxis response by E.Coli, Ca^{++} ion release from endoplasmic reticulum, oscillatory immune response of the NF-kB network and 3-stage MAPK cascade were successfully simulated.

Project CyberCell [8] uses a particle-based representation on a continuous tridimensional space. Each particle is represented as a hard sphere model of different size. This approach can simulate macromolecular crowding effect [39]; it is, however, computationally expensive.

2.4 Problematics of Cell Simulation

No current simulation methodology allows the simulation of stochastic models described with a molecular spatial resolution in a crowded state for any length of time of the order of a cell life span. The simulators are either too slow, or they are forfeiting biological details in order to reduce the processing power requirements. As described above, most simulators treat particles as dimensionless points or populations, which makes it impossible to observe crowding effects explicitly. CyberCell has demonstrated that it can have significant impacts in the behaviour of signalling pathways; it is however computationally expensive and can not support simulation the size of a whole cell yet[40]. While more efficient algorithms and the technological advances in computing power are expected to alleviate some of the performance issues, the gap is still large. The large computational cost of stochastic simulators comes from the intrinsic difference between the sequential steps of the algorithms (which runs on sequential processors) and the highly parallel nature of biological processes, where multiples molecules move, react and collide simultaneously [42].

2.5 Parallel Computing

The disparity between the sequential processing of the simulator algorithms and the parallel nature of biological processes can be reduced by designing parallel algorithms for the simulation of biological processes. Simulators such as MCell, ChemCell and CyberCell have started to explore parallelism by using multiple computers at once. MCell is also working on porting the algorithm on the Cray T3E and IBM SP supercomputers. However, this approach has not yet been thoroughly explored and it is expected that larger gains can be obtained. The increases in speed obtained with parallel processing are believed to be able to close the gap between current simulation techniques and full cell simulation.

In this section, we will explore the CA model, which has been already used to describe biological systems, although most previous work spanned only on two dimensions [52][57][54]. The CA model can be used to describe both mesoscopic and microscopic particle representations. The local interaction promoted by the CA model makes it highly susceptible to parallel implementations which negate the high computational cost of executing large systems. The CA computational modelling technique for biochemical systems is promising in terms of versatility, simplicity and scalability.

In order to exploit the parallelism offered by the CA architecture, a parallel hardware platform is required. The three most popular are currently the cluster, the GPU and the FPGA and a quick review of these three systems is presented.

Finally, some CA applications that have been successfully accelerated on parallel hardware are reviewed. We will also look at some of the most popular biological applications that have already been accelerated.

2.5.1 Cellular Automata

Cellular automata is a model studied in several scientific fields of research such as natural science, mathematics, computer science, physics, computability theory and theoretical biology [36]. It has been used to model physical and biological phenomena such as self-organisations phenomena, fluid flow dynamics (also known as lattice-gas method [15]), galaxy formation, etc. CA are also used to perform tasks such as pattern recognition, image processing, traffic simulation and parallel processing. The literature is vast, and this section introduces the basic concepts and properties of CA as well as some applications of CA in parallel processing.

Cellular automata use a discrete representation of time and space. It is a spatially decentralised model which contains a large quantity of small, simple and usually identical components connected together locally. The CA can be divided into two different parts: the *cellular space* and the *transition rule*.

The cellular space is a lattice of cells (not to be confused with the living biological cell), which are usually finite state machines (FSMs) interconnected together in a regular pattern of inputs/outputs. The lattice can contain a finite or infinite number of cells, and the boundaries, if present, can be fixed or circular. The lattice can also span over any number of dimensionalities, although for most practical applications, dimensionality varies between one and three. Each cell can be in a single state at a given time. The maximum number of states a cell can be in is often denoted by k in CA literature. The set of the k states is denoted by Σ. Each cell in the lattice is denoted by an index i and the state of cell i at time t is represented by ${s_i}^t$. At all times, ${s_i}^t \in \Sigma$. The neighbourhood ${\eta_i}^t$ of cell i is the association of ${s_i}^t$ with the states of the other cells to which cell i is connected. In one-dimensional CA, the radius r of the neighbourhood is the number of cells beyond itself that the active cell has access to in both available directions. For example, assuming an horizontal dimension, the neighbourhood of a one-dimensional CA with $r = 1$ would contain three cells: the current cell as well as the immediate left and right cell. In multidimensional lattices, the same radius r, also called range in some sources, can represent different shapes and needs to be explicitly defined. Figure 2–1 shows two different two-dimensional neighbourhoods of radius $r = 1$ a) the Von Neumann neighbourhood which has a star shape and has access to four external cells and b) the Moore neighbourhood which has a square shape and has access to the eight closest external cells. In the Lattice Boltzmann Methods terminology [62], the Von Neumann neighbourhood is called D2Q5 (for two dimensions, five directions) while the Moore neighbourhood is designated as D2Q9.

The second component of CA is the transition rule, also named the CA rule. The CA rule function calculates the next state of cell i, ${s_i}^{t+1}$, in function of the states of the current neighbourhood ${\eta_i}^t$. It is denoted as $\phi({\eta_i}^t)$.

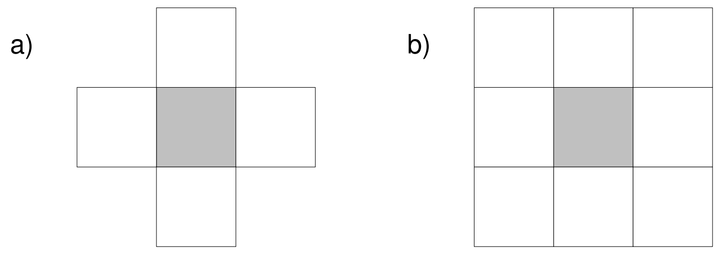

Figure 2–1: Representation of two different two-dimensional $r = 1$ neighbourhoods a) the Von Neumann neighbourhood and b) the Moore neighbourhood. The active cell being updated is shaded.

Table 2–2: One of the 256 transition rules ϕ of an ECA. In this specific case, ECA 177.

Neighbourhood	000	001	010	011	100	101	110	111
Next State	1	0	1	1	0	0	0	1

The simplest cellular automaton are members of the so-called elementary CA (ECA) [58]. ECA is the family of one-dimensional CA in which $k = 2$, $\Sigma = \{0, 1\}$ and $r = 1$. There exist 256 different ECA members accounting for the 256 different CA rules that can be created from the eight different states of the neighbourhood made of three cells. When the number of states of the neighbourhood is small, the CA rule ϕ can be described as a lookup table (LUT). Table 2–2 demonstrates an example of one of the 256 transition rule of an elementary CA. Figure 2–2 updates the states of the cells of an ECA over three time steps using the transition rule of table 2–2. The lattice boundaries wrap circularly, so that cell 0 is connected to cell 7.

One-dimensional CA are often described as a two-dimensional plane where the x axis represents the cells of the CA and the y axis represents the evolution of the cell states over time. Even the simplest elementary CA can produce complex aperiodic patterns, as can be shown in Figure 2–3.

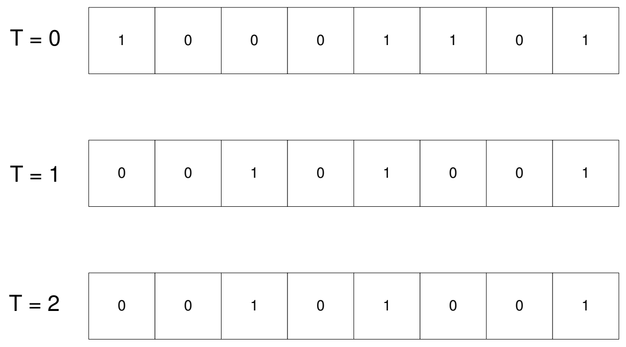

Figure 2–2: Update of the states of the ECA following the transition rule described in Table 2–2.

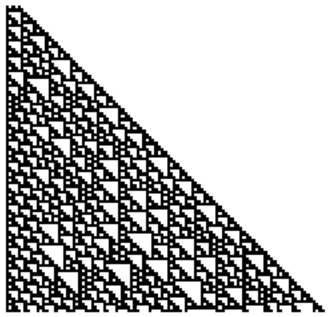

Figure 2–3: Space time diagram demonstrating the aperiodic pattern of the ECA 124. Cells in state 1 are illustrated in black while cells in state 0 are white.

Some of these patterns have even been observed in living animals and flora such as the *Conus textile* seashell.

Some notable two-dimensional, but still very simple CA include the self-replicating CA from Von Neumann [9] which, interestingly, was developed before the discovery of the principles behind the replication of DNA, but yet uses the same principles. The well-known Conway's Game of Life [5] which can create arbitrarily complex patterns, is also a good example. In the Game of Life, one can create "gliders" which are self-replicating and moving entities from which AND and OR gates can be generated. Therefore, the Game of Life can be used for universal computation. Even the simplest CA can be inherently unpredictable.

One of the strength of the CA model comes from its decentralised structure where all cells are independent of each other in that they can be updated at the same time. Thus, this decentralised model is naturally suited to be ported to parallel algorithms and architectures, and many applications have been implemented as CA for efficient computation. For example, linear time pattern recognition and efficient adders/multipliers have been achieved with simple CA algorithms. Lattice Gaz Automata (LGA) have been used to solve the incompressible Navier-Stroke equations describing fluid flows [15]. The massively parallel algorithms and the numerical stability of fixed point numbers were big advantages over the conventional approach. A triangular lattice was found to provide better results than the regular square lattice for the propagation of the particle velocities and for handling the collisions. Later, the Lattice Boltzmann Method replaced the LGA [62]. LBM uses the same general principles as the LGA, but instead of using discrete state, it uses probabilistic density functions to describe the mass, momentum and energy of particles. It retains the discrete lattice structure, discrete time and massively parallel properties and, as such, could be considered as an evolved class of CA. LBMs have also been recently used in other applications such as image processing and filtering.

Other applications such traffic simulation [53], Molecular Dynamics, Monte Carlo biological simulators [54] and many more have been implemented using CA like algorithms.

2.5.2 Hardware Acceleration

In this section, we will go through a brief overview of three of the most popular hardware architectures commonly used to accelerate an algorithm, namely, the GPU, the FPGA with reconfigurable computing (RC) and lastly, the cluster.

GPUs have a highly parallel structure with high memory bandwidth and run at high clock rate. However, they have limited flow control and data caching and are therefore best suited to stream-like applications. They can output massive amounts of work very quickly if mapped to a suitable application, and the boards are cheap. Their uses have been somewhat limited in the past due to difficult programming issues, but the recent progress of application programming interfaces (APIs) such as CUDA from NVIDIA reduced considerably the amount of time required to port an algorithm to the GPU. At this moment, GPUs have been used to accelerate several massively parallel applications in video processing, cryptography, grid computing, fast Fourier transforms and bioinformatics.

Reconfigurable computing with FPGAs has been used in a wide range of applications including signal processing, image processing, automotive, data storage, communications, aerospace and bioinformatics. By mapping the algorithms into reconfigurable logic, speedups between 10x and 100x are often obtained. They get their speed advantage from the fact that they are customised to a particular algorithm. The user can then use techniques such as pipelining and parallel execution without wasting resources to unused instruction blocks to improve the throughput of his algorithm. RC is also currently advancing technologically at a faster rate than standard microprocessors which means that observed speedups will continue to increase over time. FPGAs provide very-large-scale spatial parallelism with high internal memory bandwidth. They are very efficient at performing regular simple tasks that can be executed in parallel. Computing platforms such as the SGI RC-100 or the Cray XD1 combine CPU processing and FPGAs coupled with very fast and low latency interconnect. The FPGA is then used as an accelerator for the host processor. Two schemes can be used in this situation; the data can be streamed through the I/O ports between the FPGA and the host processor, or a chunk of data can be sent into the memory blocks of the FPGA, which is then processed until the algorithm has performed its task. Size constraints, limited external memory bandwidth and difficulty to code and debug are among the disadvantages of FPGAs. However, the regular structure of the CA algorithms had some great success in being ported on FPGA platforms [29][10]. An FPGA-accelerated biological simulator which supports three-dimensional particle movement and reaction, crowding effects, collision detection and 3D compartments has yet to be implemented.

A cluster is a group of computers linked together, usually through a local area network (LAN), all working on the same application. They are used in a wide range of applications such as web servers, databases, file servers and multiprogramming/batch processing [22]. They are most effective in applications requiring large amounts of non-unified memory and processing power and that can be divided into several parts that are not hindered by the high latency of the interconnects. The tools to port an algorithm to a cluster are also fairly well developed. Their price range can go from medium to extremely high depending on the size of the cluster. ChemCell can divide its simulation space onto several computers, and MCell is currently exploring the use of clusters to improve its performance.

2.5.3 Previous Hardware Implementations

In this section, we will look at a few algorithms which have been accelerated on parallel hardware that a) are related to computational cell biology or 2) share some similarity with a CA model.

Traffic Simulation on FPGA

While road traffic simulation is not directly related to a particle-based stochastic biological simulation, the algorithm shares a lot of similarities in organisation with the presented stochastic biological simulator. In [53], the CA algorithm TRANSIMS [47] is accelerated by using the high-level parallelism provided by an FPGA device. Using a single FPGA on a Cray XD1 supercomputer, they were able to simulate the road traffic of the entire Portland metropolitan area and achieved a 34.4x speedup over the software processor used alone.

The TRANSIMS algorithm is a CA computation on a semi-regular cell network. Each cell can hold at most one car, and the movement, acceleration and update of the cars are governed by a four-step algorithm taking into consideration the ID of the car, the states of the surrounding cells and a random number for stochastic fluctuation. All the cells are updated concurrently.

Two approaches were taken to build the system. The first one is the direct implementation, where each individual road cell is physically present in the FPGA board. This approach permitted incredible speedups which, for some cases, were larger than three orders of magnitude. The problem is that a single FPGA device could only fit a few hundred cells, while several millions are needed to represent a big metropolitan area like Portland. Thus, it was calculated that 12,400 boards would be required to map the full area, clearly an impossible implementation. The direct implementation also suffered from the difficulty to access the traffic data inside the FPGA for visualisation. The second approach is a streaming implementation where a processing engine receives a streaming flow of data as input, performs the update process, and outputs the stream of updated data back to the host processor. This implementation is scalable, as the structure of the FPGA is not related to the size nor geometry of the metropolitan area. The FPGA processing engines only process the single lane road segments of the metropolitan area, thereby removing all the complex rules and routes from the hardware processing. Fortunately, since 90% of the road cells consist of one-lane roads, most of the work is done by the FPGA. The software processor takes care of the intersections, merging nodes, routes and synchronisation of the memory. An overlap/buffer zone is used to synchronise the data between the software and the hardware processor. The streaming approach ensures that the scalability of the system is not a problem.

The CA traffic modelling has several similarities with the modelling of particles of a stochastic simulator. The update process and complexity of the structures are about the same as the presented stochastic biological simulator. The number of road cells of very large cities can be as high as 10^8, which is also not too far from the number of voxels required to describe a single cell at a molecular resolution. Also, a similar overlap/buffer zone is needed to synchronise the data between the different FPGAs and the software processor. Finally, both the throughput and bandwidth required for the two applications should be of the same magnitude. The two main differences between a biological voxel and a road cell are that nearby voxels cannot be updated concurrently and that they are operating in a 3D volume instead of a 2D area.

Biological Simulators on FPGA

While FPGAs have been used in bioinformatics in several sequencing projects such as, DNA protein string comparison and Basic Local Alignment Search Tool (BLAST) algorithms, the work on accelerating biological reaction simulators is fairly limited. Most progress has been done on the dimensionless SSA. Several teams have implemented their own version of SSA on an FPGA [42][26] device, one popular architecture being the ReSCiP [61][17]. A range of acceleration between one and two orders of magnitude is achieved. This speedup permitted the simulation of bigger systems, since even the most efficient implementation of SSA can carry a very high computational cost when used to process very large sets of stochastic simulations.

Stochastic Simulation Algorithm on GPU

While cluster and FPGA implementations of SSA have been successfully done, the GPU implementation is interesting because it offers both a shorter development time than an FPGA and is much cheaper than a cluster. In [32], the authors have shown that the SSA can be mapped very efficiently to a GPU. By mapping the Logarithmic Direct Method formulation of SSA, they have been able to achieve a speed up of 150x to 170x.

Lattice Boltzmann Method

Lattice Boltzmann Method algorithms are similar in principles to CA algorithms, one of the main differences being the cells are not limited to a specific number of states, but rather represent some density, velocity and energy continuous variables. Most principles regarding the update process are similar to CA, however, and they retain the local connectivity and concurrent updates which makes the algorithms highly parallel. Several LBM methods have been implemented in FPGAs, clusters and GPUs[62]. Successful implementations have observed speedups between one and twos order of magnitude.

2.6 SBML

Until recently, simulators and biological databases were using their own custom representation for biological models, which made the programs incompatible between each other. With the large number of different tools being developed, the lack of a universal model reduced the general efficiency of any systems biologist using these tools. A standardised description would alleviate the following problems. First, as each different tool has its pros and cons, it is not uncommon that a systems biologist has to work with several different tools at once. With different model representations, the only way to go from one software to another one is to perform a manual conversion from one format to another. This manual conversion is time consuming and error prone. Second, the use of a general model description would also remove the dependency of some models being usable only while their original software is still supported. Many valid biological models have been lost when older simulators stopped being distributed. Third, the development of a standardised model description would ensure the models could be stored in reliable databases and would be easily downloaded along with the literature published with those models. This package including both the literature and the corresponding model could provide the necessary data and terminology needed by the user to start working on the model as fast as possible. A standardised solution would also be easily expandable and able to describe increasingly complex systems [50].

The Systems Biology Markup Language (SBML) [24] was developed to address these problems. Its main goal is to improve the interoperability between softwares so that less time is spent on data formatting issues and more on actual research. It is an XML-based language which can be read and written by the programs. It is also clear enough to be readable and editable manually. Libraries parsing automatically the SBML models are provided by the SBML team and can be included in software applications.

SBML provides a basic representation of biochemical reaction networks by breaking down the simulation space into several components such as: compartment, species, reaction, parameter, unit definition and rule. SBML uses a hierarchical structure to specify the relationships between those different components. Specific annotations can be included in the model description to specify customised data.

SBML is in constant evolution and is expanding its “instruction set” by providing the new features needed by the community. The latest level included the concept of modularity. Different softwares can decide to include different modules depending on their range of applications such as rendering options, geometry, spatial diffusion, etc. Most of these new modules are currently under development. The current revision of SBML is level 3 version 1. SBML is currently being used by over 180 different software programs and this number is constantly growing.

CHAPTER 3
Algorithm for a Stochastic Particle-Based Biological Simulator

The algorithm has been designed to meet several different goals. First, as biological processes are stochastic in nature and a deterministic approach can not catch the stochastic behaviours observed in biological systems, GridCell needed to be a stochastic simulator. Second, following a similar reasoning, the particles had to follow a Brownian motion. The algorithm also had to fully support three-dimensional simulations in order to describe localisation effects, which are also expected to play a major role in cellular processes. Finally, as biological systems have been found to be "crowded" in nature and inter-particle collisions can have a huge impact on the diffusion speeds and the overall rates of reaction of the particles, GridCell was designed to be able to implicitly simulate particle collisions. This also means a microscopic representation of the particles is adopted. A simulator supporting crowding effects will show mechanics that could not be otherwise observed with other simulators or through in vitro experiments.

Three-dimensional, particle-based, stochastic, biological simulations at the molecular resolution is very computationally expensive. In fact, this kind of application is known to be too demanding to be able to simulate systems at the cellular level serially within any reasonable amount of time. So in order to circumvent this issue, the algorithm was designed so that it can be easily mapped into a massively parallel architecture.

All these goals led to an algorithm which shares similar properties to the CA algorithms used in several other applications such as traffic simulations, the LBM used in fluid simulations, error correction codes, etc. The GridCell algorithm uses a tridimensional cubic grid of type D3Q27, where each location represents a voxel which can contain a single particle. Similarly to CA algorithms, each of these voxels has only access to an immediate neighbourhood which in this case consists of the 26 surrounding voxels in addition to it own location, for a total of 27 locations. However, there is a major difference between GridCell and most CA algorithms. In CA algorithms, the update process of each cell can be performed independently from other cells. In GridCell, the neighbour particles move, collide and react with their surrounding, which creates a dependency. Fortunately, any pair of particles where the intersection of their neighbourhood is null (non existent) can not interfere with each other and can be considered independent. So, even though neighbouring particles are dependent with each other, it is still possible to divide the space into many independent areas which can be processed in parallel. Therefore, the massively parallel property of the algorithm still exists and the possibly to port the algorithm onto a parallel architecture remains very attractive.

Using a grid of type D3Q27 has several advantages; it removes the need to do complex distance calculations between the particles as the neighbours are fixed, limited in number and at already known locations. Also, by breaking the overall computation into multiple small processing elements, the logic of each of these elements is simple enough to avoid floating point calculations, which is an important feature for an efficient FPGA implementation. Finally, collisions become simply a matter of looking if the voxel is empty or not, and no expensive ray-tracing or exclusion volume algorithms are required.

The algorithm advances through time through discrete time steps where each of these time steps requires running the update process on all the particles in the simulation space. The update process of the voxels consists of two different phases: a movement phase and a reaction phase. Both phases are independent with each other in that they can be completed in any order as long as all particles are updated with both phases every time step.

From the software point of view, GridCell had to be easily usable, which led to the creation of an intuitive graphical user interface (GUI) in OpenGL. The GUI enables the user to navigate in the simulation space, start and stop the simulation, save results to various files and look at concentration and surface plots describing the system.

GridCell uses the SBML format to load the biological models to be simulated. SBML is the most popular format to store biological systems and is already used by several hundreds of other simulators, tools and databases. Supporting the SBML format allows quicker setup time and the sharing of models with other applications and databases. The SBML standard format contains information about the species, the reactions, the compartments and so on. Extra information required by the simulator, such as the simulations parameters, the shape of the compartments and the locations of the particles is added in the SBML file through custom annotations, since these features are not inherently supported in the current SBML format.

3.1 Movement Phase

The movement phase is one of the two update processes to be executed to advance to the next time step. Every particle attempts to move at every time step and every particle can move at most once per time step. As described above, each particle only has access to its immediate surrounding of 27 voxels which limits the movement in a single time step to one of these 27 nearest locations.

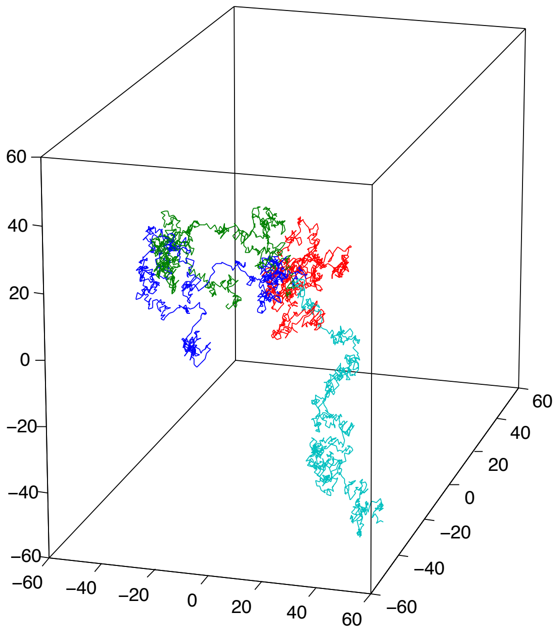

Figure 3–1: Brownian random walk

The selection of that location is made randomly, which over several time steps creates a Brownian random walk for every particles, as shown in section 3.1.1. Figure 3–1 shows an example of four different Brownian random walks of particles starting from the same location. Also, any particle attempting to move to an occupied location generates a collision. A collision prevents the particle from moving during that turn and has no effect on the other particle (ie. the other particle can still attempt to move if it has not tried to move yet). The order in which the particles are updated does not have any significant impact on the overall simulation results, as the difference in behaviour a different ordering could have in the system is contained within the inherent stochasticity of the simulator. Changing the update order has an effect similar to using a different starting random number generator “seed”.

3.1.1 Diffusion

In cell biology, diffusion is the main type of movement for the particles. In GridCell, diffusion is achieved through the movement phase, which moves every particle in a random direction at every time step. Particles following a Brownian random walk also follow the well-known Einstein-Smoluchowski equation

$$< r^2 >= 2dDt, \tag{3.1}$$

where $< r^2 >$ is the mean-square displacement, d is the dimensionality, D the diffusion coefficient and t the elapsed time. Movement in GridCell should also follow a similar behaviour. Figure 3–2 shows the mean-square displacement in units of voxel $< v^2 >$ (averaged over 1000 iterations) versus the number of time steps n_{ts} when the probability of movement of the particles at every time step is equal to one. The mean-square displacement $< r^2 >$ is equal to the mean-square displacement of voxels $< v^2 >$ multiplied by the side length squared of the voxels s_{vox}^2. Similarly, the elapsed time t is equal to the number of time step elapsed n_{ts} multiplied by the length of each time step l_{ts}. In agreement with the Equation 3.1, the mean-square displacement $< v^2 >$ increases linearly with the number of time steps n_{ts}. This leads to the following linear relation

$$< v^2 >= An_{ts}, \tag{3.2}$$

where A, the slope of the graph, is in units of meter^2 per second. By substituting $< v^2 >= \frac{<r^2>}{{s_{vox}}^2}$ and $n_{ts} = \frac{t}{l_{ts}}$, we get

$$< r^2 >= \frac{A{s_{vox}}^2}{l_{ts}}t = 2dDt. \tag{3.3}$$

Since the probability of movement at each time step of the particle is equal to one, D can be substituted for the maximum diffusion speed D_{max} supported for a given time step and voxel size. This upper limit on diffusion speed is caused by the design decision of restraining particle movement to its immediate neighbourhood (the D3Q27 grid). In the case where no such limit exists, the diffusion equation is solved by a tridimensional Gaussian probability density function. By calculating the slope of the graph and setting the dimensionality d equal to 3, D_{max} can be calculated as

$$D_{max} = 0.335{s_{vox}}^2/t_s. \tag{3.4}$$

Smaller diffusion speeds are simulated by applying a different probability of movement such that

$$D = p_m D_{max}, \tag{3.5}$$

where p_m is the probability of movement of a particle at every time step. As long as the diffusion speeds of the particles are smaller than D_{max}, diffusion is modelled correctly. If a larger diffusion speed is needed, one can reduce the time step or increase the size of the voxels.

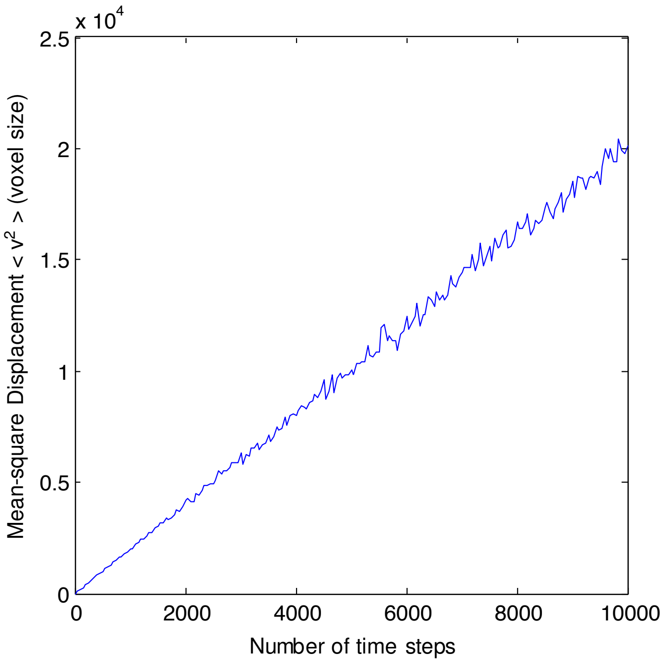

Figure 3–2: Diffusion in GridCell

3.2 Compartments

In cell biology, the term compartment is generally used to describe all the parts, located inside a membrane, usually made of lipid, which have a specific function inside a larger organism. The membrane forming the compartment can be solid, semi-porous or porous. More generally, the term compartment can also describe any structure used to add spatial organisation to a model. GridCell's compartments fit both descriptions.

Every voxel has a compartment field representing the compartment the voxel is in. Different compartments are represented by different values. The main effect of compartments is to change the diffusion speed of the particles moving through or inside them. Instead of having a single diffusion speed, which translates into a probability of movement p_m per time step as mentioned in section 3.1.1, a system containing N different compartments generates a NxN matrix for each species containing every combination of movement going from compartment A to compartment B. Each entry $[A][B]$ of the matrix represents the permeability rate or the probability of entry $p_{m_{AB}}$ of going from compartment A to compartment B. Note that when A and B are the same value, the entry represents the probability of movement of the species moving inside the given compartment. $p_{m_{AB}}$ having a different value from $p_{m_{BA}}$ can be used to represent an asymmetric flux between two different compartments, which could be caused by some kind of active transport. By adjusting the various diffusion speeds and permeability rates, it is possible to let a particle move free, restrain it, or even block it completely from entering, exiting or moving inside any given compartment.

Reactions with particles belonging to different compartments are prohibited. If needed, one can simulate a reaction occurring at the surface of a membrane by adding an extra thin compartment surrounding the inside compartment.

The probability of movement matrix is the only memory element in the algorithm scaling with the square ($O(n^2)$) of a parameter. This could potentially lead to some scaling issues. Fortunately, the parameter in cause, the number of compartments, is usually a very small number. Most available SBML models have between one and three compartments. More complex models, showing localisation effects could have around 10 compartments. Also, for the very complex systems, the moving ratio matrix is expected to be sparse, as several combinations of compartments would be invalid. For example, if compartment A is inside compartment B, which is inside compartment C, then A and C are not in contact with each other, which means that any diffusion speed assigning A to C or C to A is irrelevant. It is also assumed some compartments would restrict access to some type of particles, which further increase the number of zero entries in the matrix. For very large system simulations, a sparse matrix representation can be used.

3.3 Reaction Phase

The reaction phase is the second update process required to complete a time step. Similarly to the movement phase, a particle may react only once per turn and only with its immediate surrounding. It also shares the same properties regarding the ordering of the update. The purpose of the reaction phase is to simulate the common interactions between the particles in biological systems. These interactions include aggregation events such as molecular complex formation/dissolution, or conversion events such as chemical reactions. Some of these reactions can involve a large number of reactants and products. Fortunately, it is usually possible to decompose those reactions into a cascade of several small reactions. Therefore, only the simplest reactions involving three or less participants are directly supported by the simulator. Larger reactions involving more than three particles are automatically decomposed into several elementary reactions by the algorithm. The procedure is described in Section 3.3.2. The probability of reaction per time step is derived from the overall rate of reaction, the voxel size and the length of the time step. It is very similar to the approach taken by ChemCell [37] and is derived from the macroscopic law of mass action. There are three different reactions involving three or less participants: one reactant and one product, one reactant and two products and two reactants and one product.

3.3.1 Simple Reactions

Let's consider the two types of reaction involving a single reactant

$$A \rightarrow B \tag{3.6}$$

$$A \rightarrow B + C. \tag{3.7}$$

Both reactions have a forward rate of reaction k in units of time^{-1} and a time step of t second. Assuming a well-mixed approximation and N particles of type A in the system, then in both cases, the expected number of reactions per turn is given by $N(1 - e^{-kt})$. Each individual particle has a probability equal to $1 - e^{-kt}$ of reacting every time step. In our stochastic model, a uniform random number R_n between 0 and 1 is generated for each particle, and the reaction occurs if $R_n < 1 - e^{-kt}$.

In the reaction with only one reactant and one product, the reactant is simply replaced by the product. In the reaction with one reactant and two products, a search is first conducted in the surrounding area. If at least one free voxel in the surrounding area of the particle is found, the reaction takes place, and the second product is positioned in the free location while the first product is placed at the position of the initial reactant. The reaction is blocked if no free position is found. Simulations have shown that this limitation starts affecting the overall reaction rate of the reaction when more than 98% of the voxels of the simulation space is filled with particles. A system containing such a high particles per voxel ratio is so crowded that it becomes almost static and, therefore, of limited interest.

Consider the following reaction with two reactants:

$$A + B \rightarrow C, \tag{3.8}$$

with a rate constant k in units of (molarity*time)$^{-1}$ and a time step of t second. Assuming N_a particle of type A, N_b particle of type B, a Volume V and the Avogadro's number A_v, which refers to the number of molecules ($6.022 * 10^{23}$) contained in one mole of matter, then the total number of reactions N_r in a well-mixed system is given by

$$N_r = \frac{kN_aN_bt}{A_vV}. \tag{3.9}$$

On average, in a well-mixed situation, the desired number of reactions in GridCell should be equivalent to the result of the above equation. In our system, particles can only react with their immediate surrounding locations. In a well-mixed system, the number of A,B pairs that are close enough to each other to generate a reaction is given by $N = N_a N_b V_c / V$ where V_c is the volume of the cube containing the 27 neighbouring voxels and V is the total volume of the simulation. If each of those pairs react with probability P, then $N_r = NP$. Setting the two equations $N_r = NP = kN_a N_b t/(A_v V)$ gives the equation

$$P = \frac{kt}{A_v V_c}. \tag{3.10}$$

The formula is independent of V, N_a and N_b as expected. Also, for a given rate constant k, it is possible to have a set of parameters t/V_c such that P is greater than 1. If that is the case, a smaller time step or larger voxel volume has to be selected. A random number R_n is generated. If $R_n < P$, then the first reactant searches its surrounding area for the second reactant. If it is found, the reaction takes place and the product is placed at the location of the first reactant. If no reactant is found, the reaction is aborted. Note that only one of the two reactants attempts to react with the other one, doing so makes it possible to get the right overall rate of reaction while reducing the total number of operations needed to complete the reaction phase.

When the same species participates in more than one reaction, the probabilities of reaction P are added one after the other, and the value of the random number R determines which reaction is selected. For example, if species A participates in reaction R_1 and R_2, with probability P_1 and P_2. Then, reaction 1 is selected if $0 \leq R < P_1$, reaction 2 is selected if $P_1 \leq R < P_1 + P_2$ and no reaction takes place if $R \geq P_1 + P_2$. This approach is very similar to the decision process performed in the SSA to select which reaction takes place at every time step. The time step needs to be small enough to make sure that the sum of the probabilities is not larger than one.

When two reactants of the same species form a product like

$$A + A \rightarrow B, \tag{3.11}$$

the individual rate of reaction of particle A needs to be modified to ensure that the overall rate of reaction is respected since there is no way to avoid the fact that each reactant will attempt to react with the other one. Assuming y is the overall probability of reaction and x is the individual probability of reaction of species A, then

$$x = 1 - \sqrt{1 - y}. \tag{3.12}$$

3.3.2 Large Reactions

Complex reactions are implemented by creating a cascade of several elementary reactions. This process, done automatically by the software, breaks the complex reactions into a series of simpler reactions by introducing temporary species. For example, consider the following reaction with one reactant and five products

$$A \rightarrow B + C + D + E + F, \tag{3.13}$$

where k is the rate of reaction in time^{-1}. For each product exceeding two, a temporary species is created. In this specific case, three temporary species are created. The reaction is then broken down into:

$$T_1 \rightarrow B + C, \tag{3.14}$$

$$T_2 \rightarrow D + E, \tag{3.15}$$

$$T_3 \rightarrow F + T_1, \tag{3.16}$$

$$A \rightarrow T_2 + T_3, \tag{3.17}$$

where T_1, T_2 and T_3 are respectively the first, second and third temporary species. By setting the rate of reaction of equation 3.17 equal to k and the probability of reaction of equations containing any temporary species on the reactant side to one, we reduce the artefacts caused by the creation of the temporary species to a minimum. Indeed, the temporary species disappear from the system as quickly as possible and the overall rate of reaction is identical.

Shown below is an example where more than two reactants merge into a single product:

$$A + B + C + D + E \rightarrow F. \tag{3.18}$$

The procedure is similar to the previous case; a single temporary species is created for each reactant above two:

$$A + B \rightarrow T_1, \tag{3.19}$$

$$C + D \rightarrow T_2, \tag{3.20}$$

$$E + T_1 \rightarrow T_3, \tag{3.21}$$

$$T_2 + T_3 \rightarrow F. \tag{3.22}$$

In order to obtain the same overall probability of reaction and to reduce the impact of the temporary species on the system to a minimum, the probability of reaction of any reaction containing temporary species on the reactant side (equations 3.21 and 3.22) is set to one. Assuming that P is the probability of reaction of the reaction presented in equation 3.18 and P_1 and P_2 are the probability of the first and second simple reactions $A + B \rightarrow T_1$ and $C + D \rightarrow T_2$, then we set $P = P_1 P_2$. We also set $P_1 = P_2$. Equating the two equations gives $P_1 = P_2 = \sqrt{P}$. In general, the probability of the simple reactions P_n containing no temporary species is equal to

$$P_n = P^{\frac{1}{\lfloor \frac{N_{reactants}}{2} \rfloor}}, \tag{3.23}$$

where P is the overall probability of reaction and $N_{reactants}$ is the number of reactants of the initial reaction.

The two procedures are superposed together when handling reactions with several reactants and products.

Each temporary particle has a parameter $lifetime$ indicating the number of turns the particle has to live in the system before reverting back to its previous state. Similarly to the case of the simple reaction with one reactant and two products, if no room is found, the reaction is aborted and the temporary species will attempt to revert back to its previous state during the next time step. Simulation have shown that the overall rate of reaction is not affected until the ratio of occupied voxel becomes larger than 98%. It is assumed that biologically relevant model operates below this threshold. The short lifetime of temporary particles is important for two reasons. First, it makes sure temporary particles are effectively temporary and never stay in the system for long periods of time. It also makes sure that all the reactants are close to each other for a reaction to complete. Usually, a lifetime between two and three time steps is reasonable since it gives enough time to react with the neighbouring particles while making sure temporary particles do not constitute the bulk of the system.

3.3.3 Reversible Reactions

Reversible reactions are handled by creating two different separate reactions, one for the forward reaction with the forward reaction rate and one for the backward reaction with the corresponding backward reaction rate. The following reaction

$$A + B + C + D + E \leftrightarrow F, \tag{3.24}$$

Table 3–1: GridCell performance versus system size.

Number of Voxels	$1e3$	$1e4$	$1e5$	$1e6$
Number of Particles	$3e2$	$3e3$	$3e4$	$3e5$
Time (s)	$1.62e-4$	$1.58e-3$	$1.6e-2$	$1.7e-1$

Table 3–2: GridCell performance versus number of voxels. Zero particles.

Number of Voxels	1e3	1e4	1e5	1e6
Time (s)	1.6e-5	1.6e-4	2.14e-3	2.06e-2

where forward reaction rate is k_f and backward reaction rate is k_b, is then split into

$$A + B + C + D + E \rightarrow F, \tag{3.25}$$

with a reaction rate k_f and

$$F \rightarrow A + B + C + D + E, \tag{3.26}$$

with a reaction rate k_b.

3.4 Performance Analysis

Tests have been conducted to determine how the software reacts to different system sizes. The tests have been executed on a stand-alone microprocessor: a 3.2 GHz P4 with 2GB of RAM. The algorithm is computed serially. As it can be shown in Table 3–1, the time required to compute a time step increases linearly with the number of particles and voxels present in the system. Tables 3–2 and 3–3 demonstrate how the performance is affected by independently modifying the number of voxels or the number of particles. It can be seen that both parameters affect performance. Thus, an occupied voxel takes more time to process than an empty voxel and an empty voxel takes more time to process than no voxel at all. The maximum number of particles that can be currently simulated is equal to the maximum number of voxels that can be supported, which is 10^7. Table 3–4 shows that the number of reactions occurring at each time step has a negligible effect on the performance. The reason is that all reactions have to be tested, regardless of whether or not they actually react. There are no practical limitations to the total number of chemical species or the number of different reactions supported by the software.

Table 3–3: GridCell performance versus number of particles. 10^6 voxels.

Number of Particles	1e3	1e4	1e5	5e5
Time (s)	21.3e-2	26.4e-2	68.1e-2	22.8e-1

Table 3–4: GridCell performance versus the average number of reactions.

Average Number of Reactions	0	16.5e3	29.5e3	39.5e3	47.5e3
Time (s)	7.4e-2	7.3e-2	7.25e-2	7.2e-2	7.1e-2

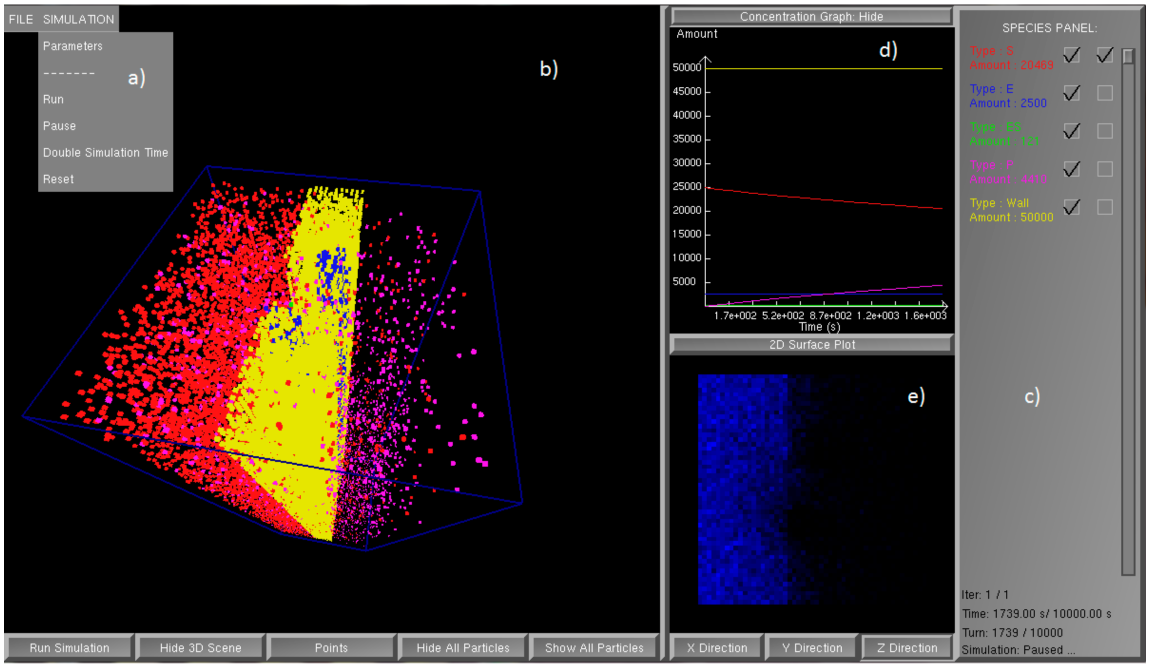

Figure 3–3: GridCell user interface

3.5 User Interface Features

The rendering is implemented in OpenGL, and most user-interface functions are written using the PLIB library, available online (http://plib.sourceforge.net/). The user interface is shown in Figure 3–3 and consists of a) a menu system, b) an interactive 3D simulation space, c) a species panel, d) a 2D plot of concentration versus time, and e) a 2D plot of concentration versus space.

The menu system (Figure 3–3a) provides the ability to load SBML models, set parameters and control the simulation. User-designated simulation parameters include the number of times to run the simulation, the time step, the total simulation time and the update rate, which is the frequency of the refresh rate of 2D graphs and 3D visualisation, as well as the frequency at which the results are saved to file. These preferences may be entered in the configuration panel or can be stored as annotations in the SBML file itself. GridCell computes the means and the standard deviations of the concentration of the species over time if the user chooses to run multiple iterations of the simulation. The particle concentrations and the 2D surface plot data are saved in user-specified tab-delimited files.

A key feature of the GUI is the ability to interact with the three-dimensional simulation volume (Figure 3–3b). Users can navigate into the 3D scene with mouse and keyboard controls to rotate, pan and zoom. Buttons are present to i) start/pause simulations, ii) change the particle representation from cubes to points for faster rendering, iii) turn off the visualisation for optimal simulation performance, and iv) hide or show all particle types.

The species panel (Figure 3–3c) contains the current amount of each species, and allows species selection for the visualisation plots. A second column specifies which species to render in the 2D surface plot of concentration versus space (Figure 4e). Particle colours are automatically selected from a predefined colour palette.

Finally, two plots to summarise particle concentrations with respect to time (Figure 3–3d) and space (Figure 3–3e) are provided in real-time. The 2D spatial plot displays increasing concentration with increasing brightness along a selected Cartesian axis.

3.6 SBML Support

GridCell uses the SBML format to describe its model. SBML models are inherently non-spatial, as most tools were not including spatial effects when it was first developed ten years ago. The latest revision has plans to include modules providing spatial information into the models; however, the SBML spatial specification has not yet been completed. As a result, many details such as the simulation parameters, the shape and location of the compartments as well as the diffusion speed and location of each species inside the different compartments can not be specified in the current SBML format. This information is entered by adding specific annotations to the various entities that need to be updated. Default values are used if no specific parameters are entered.

Model Annotations

The simulation parameters such as the number of times to run the simulation, the time step, the total simulation time, the update rate and the voxel volume can either be entered through the UI or be assigned in the SBML file inside the model annotation. The *parameter* argument line inside the model entity is used to define the values of the five simulation parameters. The *concentration* and *localisation* argument lines specify if the data is to be stored and the name of the file where the data is saved. The format and the default values are presented below.

```
<parameter numberOfIteration="1" timestep="0.001" length="10" updateRate="1"
   voxelVolume="3.2768e-23"/>
<concentration saveResults="yes/no" fileName="conc.txt" />
```

```
<localization saveResults="yes/no" fileName="loc.txt" />
```

Compartment Annotations

Compartments are created in the simulation in the same order as they appear in the SBML file. The first compartment is always the largest and represents the total simulation space. Its shape can be modified, but not its relative location. Compartments created afterwards overwrite previous compartments if they overlap, which is used to create compartments inside others compartments. The shape of compartments and their locations can be individually specified inside their respective annotation. The *shape* argument specifies the proportion of each side of the rectangular box forming the compartment. The overall volume stays constant regardless of the scale of the values used (plus or minus some possible quantisation errors). The *location* argument specifies the position of the centre of the compartment to the overall simulation space (i.e.: the first compartment). For example, if x,y,z values are set to 0.5, the centre of the compartment is located at the centre of the simulation space. Finally, a new compartment can be assigned to be of same type of a previously created compartment with the *type* argument line and the name of the previous compartment. This last option enables the creation of complex shapes as well as the possibility of having multiple compartments of the same type at different locations. As an example, here are the three arguments presented with possible values.

```
<shape x="1" y="1.3" z="0.45"/>. Where X,Y,Z are floating point values greater than 0.
   (Default: x=1 y=1 z=1)
<location x="0.2" y="0.7" z="0.666"/>. Where X,Y,Z are values between 0 and 1.
   (Default: x=0.5 y=0.5 z=0.5)
<type compartment="nameOfCompartment"/> (Default: No default)
```

Species Annotations

The diffusion speed of each species when entering, exiting and moving inside a compartment can be specified through the *diffusion* annotation line. Supported units are cm2/s if *prob* = “*no*”, or probability of movement per time step if *prob* = “*yes*″. Diffusion speed is assigned from *compartment*1 going into *compartment*2. If *compartment*1 is equal to *compartment*2, then the diffusion speed when moving inside the given compartment is assigned. There is no maximum to the number of different diffusion speeds that can be entered per species. However, writing an entry with the same compartments twice will overwrite the initial entry. The location of particles inside a compartment can be specified through the *location* annotation. The coordinates represent the fraction of the compartment where the particles are randomly distributed and is specified with the arguments *xmin*, *xmax*, *ymin*, *ymax*, *zmin* and *zmax*. The *frac* parameter represents the fraction of the particles assigned to the compartment placed in the specified location. There is no limit to the number of *location* lines that can be entered but the sum of the *frac* should not exceed one for each species in the same compartment.

```
<diffusion compartment1="comp1" compartment2="comp2" speed="0.02" prob="yes/no"/>
   (Default: compartment1="0" compartment2="0" speed="1" prob="yes")
   (all other entries are set to 0)
<location xmin="0.0" xmax="0.5" ymin="0.5" ymax="0.9" zmin="0.3" zmax="0.6" frac="0.8"/>
   (Default : xmin="0" xmax="1" ymin="0" ymax="1" zmin="0" zmax="1" frac="1")
```

The same species (same name must be used) can be placed in different compartments.

3.7 Initialisation of the Simulation Space

The software can be divided into three different parts: the graphical user interface, the initialisation of the simulation space and the update process. The GUI and the update process have both been described earlier in this chapter. The task of the initialisation phase is to build the simulation space and precompute the databases required by the update process to perform its task without interruptions. The initialisation phase is ran once at the beginning of the simulation and, once completed, the update process can iterate as many times as needed on the simulation space. Its main input is the SBML file, which is parsed to get the relevant data. The initialisation also issues warnings and errors when encountering invalid parameters. Most warnings come from a combination of parameters which yield a value for a rate of reaction or a rate of movement larger than one. Errors are triggered when invalid parameters completely prohibit the creation of the simulation space. For example, trying to create a system that exceeds the maximum size, generating more particles than the number of available voxels or generating out of bounds compartments.

The first step of the initialisation phase is to parse the unit definition section of the SBML file. The software has to know which units are used to represent the volumes, concentrations and diffusion speeds of the various elements of the system in order to do the correct calculations.

The next step is the creation of the compartments. The initialisation starts with the main compartment, which corresponds to the full simulation space, and constructs the other compartments in the order that they appear in the SBML file. The size, shape, type and position parameters are used to create the compartments accordingly. Currently, only rectangular shapes are supported, although more complex shapes can be created by superposing several compartments of the same type together. Since the compartments are created in the initialisation phase, which is done in software, arbitrary complex shapes could be supported without modifying the update algorithm.

Once the compartments are created and the voxels are initialised, the species database is built. Each new species is given its own entry, and species already entered in the database are merged into their corresponding entry. The particles of each species are then created in the simulation space according to the amount, the location and the compartment values specified in the SBML file. The diffusion speeds of the particles are also parsed and converted into the probability of movement per time step during this stage.

Finally, the last step is the creation of the reaction database. For each reaction in the SBML file, the software looks at the number of reactants, products and their corresponding stoichiometry and, if needed, decomposes the reaction into several smaller reactions by introducing temporary species, as described earlier in this chapter. The mathematical expression describing the kinetics of each reaction is parsed in order to calculate the probability of reaction per time step. If the reaction is reversible, another set of reactions is generated, but with the products as reactants and the reactants as products.

3.8 Results and Discussion

In order to validate the algorithm, several molecular models were simulated and their results compared to an already well-known and proven approach, the SSA from Gillespie [18]. It is a very good candidate to validate our model as both approaches should provide equivalent results for well-mixed systems. The first model is a simple reversible reaction $A + B \leftrightarrow C$. The second one is a Michaelis-Menten system, which describes the kinetics of many enzymes. The third example demonstrates the effects of crowding by adding a large amount of non-interacting particles to a Michaelis-Menten system. The fourth example introduces how spatial constructs and localisation effects can play an important role in the overall behaviour of the system. The last example simulates the ultra-sensitivity of a 3-stage mitogen-activated protein kinase (MAPK) cascade, which has also been observed experimentally in Xenopus extracts.

3.8.1 Simple Reaction

This system is a simple reversible reaction involving three different species A, B and C in the following manner: $A + B \leftrightarrow C$. The forward reaction $A + B \rightarrow C$ has a rate of reaction k_f of 10^{10} per mole per second. The reverse reaction $C \rightarrow A + B$ has a rate of reaction k_b of 1 per

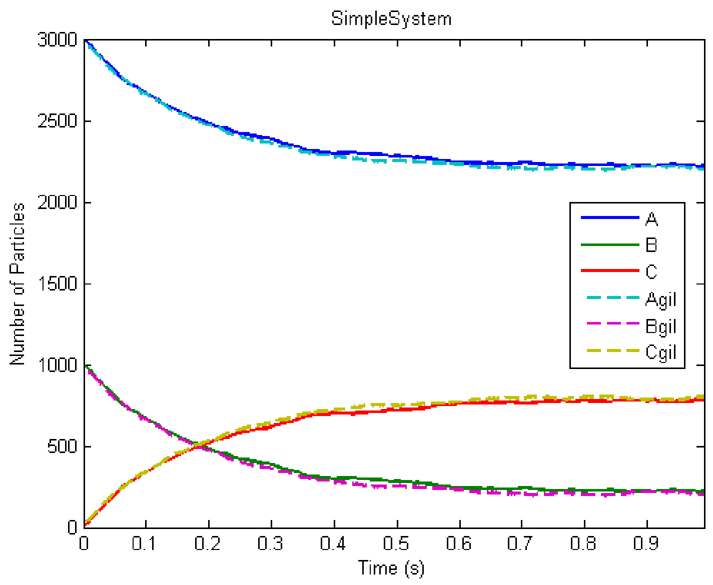

Figure 3–4: Comparison between GridCell and SSA for the reaction $A + B \leftrightarrow C$.

second. The simulation space is a cube with a volume of 10^{-11} litres and the time step is 10^{-4} seconds. The initial number of particles is 3000 A particles, 1000 B particles and 0 C particles. The system reaches steady state fairly quickly and after 1 second of simulated time, the species reach equilibrium with the exception of some small stochastic noise. The SSA simulator has been set with similar parameters. The results are shown in Figure 3–4. Both simulators produce the same results with small, but expected stochastic fluctuations. This model has also been simulated by the ChemCell software with similar results [37]. These results support the idea that the discretisation of the volume into a grid does not affect system behaviour under these conditions.

3.8.2 Michaelis-Menten Reaction

The Michaelis-Menten equations are used to describe most enzymatic reactions. Its kinetics is given by the following equation:

$$E + S \leftrightarrow ES \rightarrow E + P. \tag{3.27}$$

The enzyme E reacts with the substrate S to form the complex ES with a rate of reaction k_1. ES decomposes into the enzyme E and a new product P with a rate k_2, or reverts back to its original form $E + S$ with rate k_r . As in the previous case, the simulation takes place in a cube of 10^{-11} litres, the number of enzymes E is 1000 particles and the initial amount of substrate S is 3000 particles. The forward rate of reaction k_1 of $E + S \rightarrow ES$ is 10^{10} per mole per second and

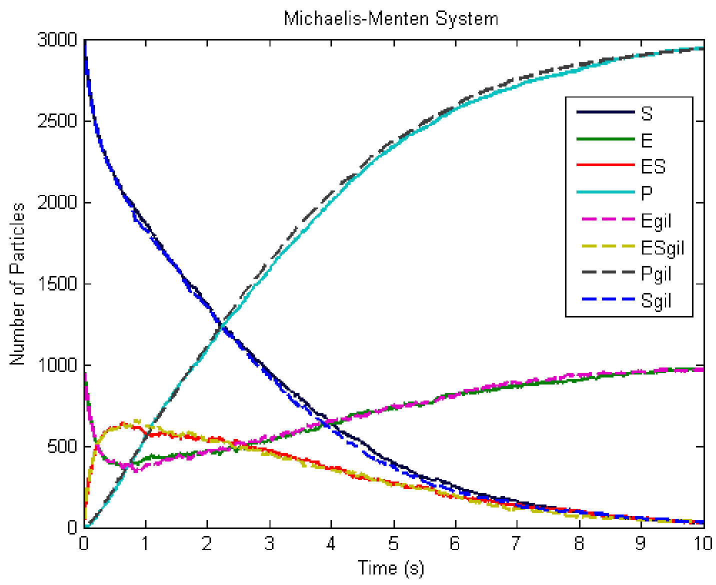

Figure 3–5: Comparison GridCell and the dimensionless SSA for a Michaelis-Menten system.

the reverse rate of reaction is 1 per second. The forward rate of reaction of $ES \rightarrow E + P$ is also 1 per second. The simulation runs for 10 seconds, and the time step in both cases has been set to 10^{-3}. The results are compared to the SSA algorithm and presented in Figure 3–5. Similarly to the previous test, both approaches produce comparable results.

3.8.3 Crowding

One of the main differences between GridCell and other simulators is its ability to simulate crowding effects. Molecular crowding occurs when the particle density affects movement and reactivity. Crowding is typically ignored in most models since kinetics are often based on controlled, in vitro conditions that are not crowded. In addition, simulators do not typically support this feature because it is computationally expensive to keep track of all particle positions and their excluded volume, and to implement collision-detection algorithms. Some simulators (e.g. Smoldyn [3]) have shown crowding effects by explicitly introducing cubic obstacles [34] in the model. In contrast, GridCell implicitly exhibits molecular crowding effects by allowing inter-particle collisions. We demonstrate the effect of crowding by adding inert particles to a Michaelis-Menten system. Inert particles do not react with other molecules, but their presence reduces the amount of available space, impacting the diffusion speed and affecting the overall number of reactions of the active particles. The simulation parameters are described in Table 3–5.

Figure 3–6 shows the number of products over time for a wide range of concentrations of inert particles averaged over 20 iterations. The individual simulations provided almost identical

Table 3–5: Simulation parameters for the crowded system.

Volume (litres)	10^{-14}
Number of S particles	3000
Number of E particles	1000
k_1 $(M^{-1}s^{-1})$	10^7
k_2 (s^{-1})	1
k_r (s^{-1})	1
Simulation time (s)	10
Timestep (s)	10^{-3}

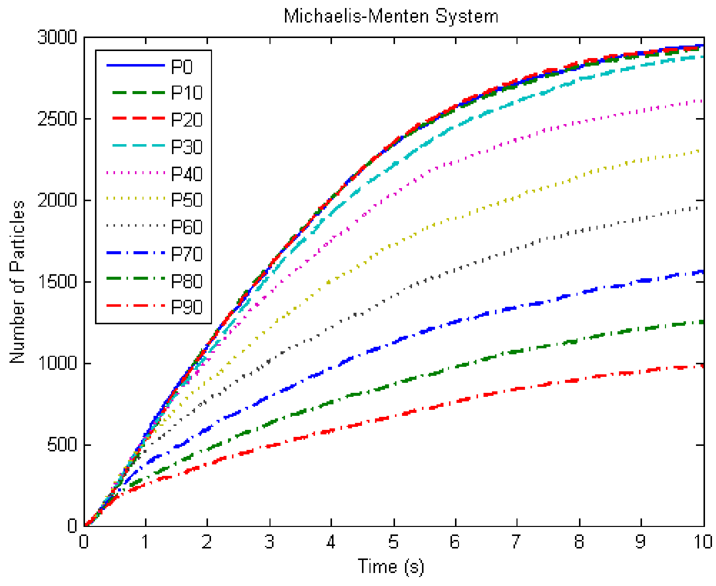

Figure 3–6: Effect of crowding on Michaelis-Menten product formation using GridCell.

results compared to one another with a relative standard deviation smaller than 3.5% at the end of the simulation. The number indicated in the legend is the percentage of voxels occupied by inert particles. In this specific example, with a voxel size of 3.2^{-20} litres, this amounts to approximately 30000 inert particles per step of 10%. Interestingly, the maximum rate of reaction is obtained when the inert particles occupy 20% of the volume, confirming the fact that macromolecular crowding may enhance reaction rates, as the particles have to search a smaller volume to find each other[63]. However, above 30%, the reaction rates decrease linearly as more and more inert particles are added. Interestingly, the rate of reaction for the first half second is roughly the same regardless of the amount of crowding in the system. This can be explained by the fact that at the beginning of the simulation, a roughly equal number of enzyme-substrate pairs are close enough to each other to be able to react. It is only after the initial set of pairs have reacted and after the substrate and the enzyme particles have to move to form new pairs that an effect is observed, as the movement is hindered by the crowding of the system.

3.8.4 Localisation

Localisation of particles, either by recruitment to a specific location or by anchoring them in structured environments, is expected to affect cellular processes. Here, we examine the effect of localisation on reaction rates when a system is not well-mixed. Localisation to cellular structures such as membranes may influence the overall behaviour of the system by fixing position, reducing diffusion and hence affecting the rate of collision between interacting particles. The biochemical model is a Michaelis-Menten reaction where enzymes are localised to regions of a semi-porous membrane made of immobile inert particles. The substrate particles are also all placed on one side of the membrane. This example is similar to the one presented in [20]. The top view of the structure is shown in Figure 3–7. Substrate particles initially located on the left side slowly migrate to the right side, as shown in Figure 3–8. The S concentration is still much higher on the left side after 10s of simulation time. Concentration of the S is lowest at the two enzyme sites, since the particles are converted to products when interacting with enzymes embedded in the membrane. Figure 3–9 shows the evolution of the species P. Figure 3–10 shows the difference in the overall reaction rate between a well-mixed system and a system with the structure described by Figure 3–7. Both simulations have the same number of particles, the same volume and the same reaction rates. However, the overall speed of reaction is substantially different between the two systems. Due to the presence of the semi-porous membrane and only two specific areas where the reaction can take place, the non-well-mixed system exhibits a much slower reaction rate than the ideal well-mixed

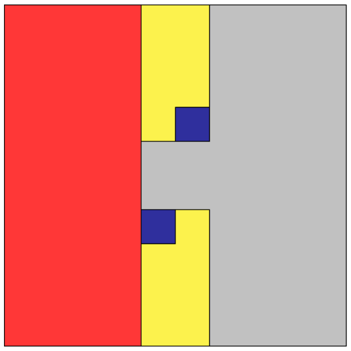

Figure 3–7: Top view of the simulation structure. Red area indicates location of the substrate, blue areas indicate the locations of the enzymes and yellow areas indicate the inert particles forming the porous membrane.

case. This demonstrates that the structure can have a significant impact on the behaviour of a biological pathway and that the well-mixed assumption can produce very important errors.

3.8.5 3-Stage MAPK Cascade

The family of MAPK pathways has received a lot of attention from biologists both in experimental studies and in computational cell simulations. While the biological responses of the cascade are highly varied and a full description of the processes is not within the scope of this thesis, the general structure of the cascade remains similar for most cases. The pathway starts by reacting to an external stimuli, usually from the outer plasma membrane. The cascade is used to relay this outside information inside the cytoplasm and nucleus to regulate various intra-cellular activities. The external stimuli activates the first stage of the chain consisting of MAPK kinase kinase (MAPKKK) which will then promote the activation of the second stage made of MAPK kinase (MAPKK), which will finally activate the final stage consisting of MAPK. The concentration of activated MAPK is the output of the cascade.

In this example, the chain starts with the MAPKKK which is activated by an external stimuli, the enzyme $E1$. The activated MAPKKK (denoted by MAPKKK*) then promotes the phosphorylation of the MAPKK into MAPKK-P and then in turn promotes the phosphorylation of the MAPKK-P into MAPKK-PP. Finally, the doubly-phosporylated MAPKK-PP promotes phosphorylation of the MAPK into MAPK-P and the MAPK-P into the MAPK-PP. The concentration level of the MAPK-PP is the output of the chain. Figure 3–11 describes the 3-stage MAPK cascade. More details can be found in [28].

[23] were intrigued by the fact that the MAPK signalling pathway cascade arrangement had three different stages, while a number of other membrane to nucleus pathways were a protein cascade

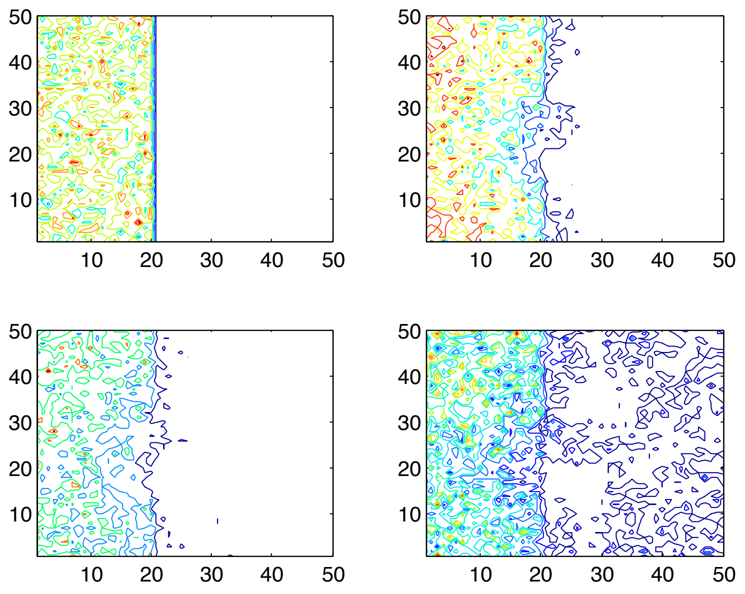
Figure 3–8: Top view contour plot of substrate concentration at $t = 0$, $t = 2$, $t = 6$ and $t = 10$.

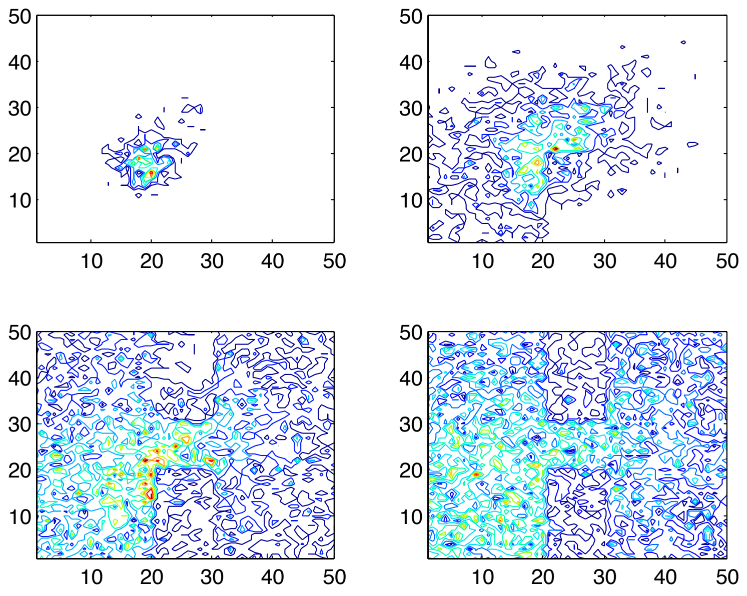
Figure 3–9: Top view contour plot of product concentration at $t = 0$, $t = 2$, $t = 6$ and $t = 10$.

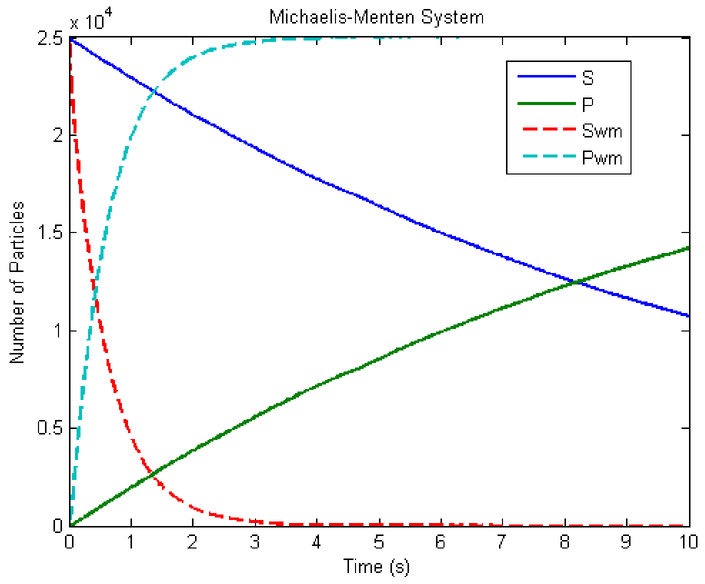

Figure 3–10: Concentration of the same Michaelis-Menten system under two different conditions. S and P consider localisation effect, Swm and Pwm assume a well-mixed solution.

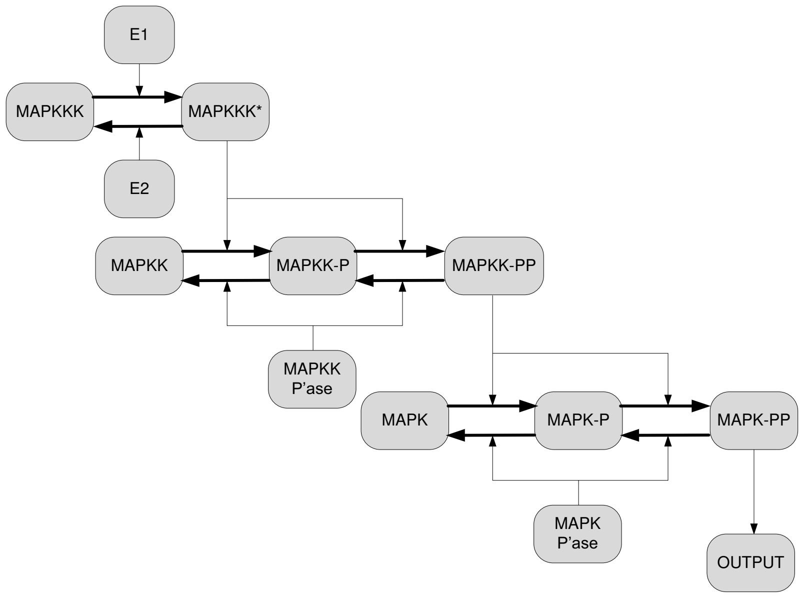

Figure 3–11: Schematic view of the MAPK cascade from [23]. MAPKKK is activated by E1 (the input) into MAPKKK*, which stimulates the MAPKK to phosphorylise twice. The doubly phosphorylated MAPKK-PP promotes the phosphorylation of the MAPK and MAPK-P protein. MAPK-PP is considered the end of the chain and the output of the system. P'ase denotes phosphatase and promotes the dephosphorylation of the kinases.

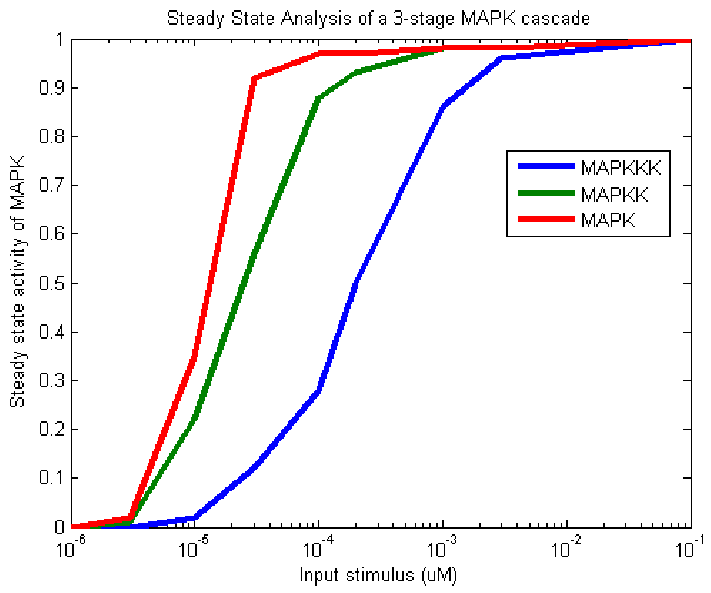

Figure 3–12: Steady state response of the 3-stage MAPK cascade.

with a single stage. Also, the fact that each stage needed to be phosphorylated twice should have an important incidence on the overall dynamics of the system. It was found, both numerically and experimentally, that the 3-stage cascade exhibits an ultra sensitive behaviour denoted by a large Hill coefficient varying between 4 and 5. The Hill coefficient is a measure to determine the amount of cooperativity in a binding process. A value larger than one demonstrates an ultrasensitivity response compared to the response curve of a Michaelis-Menten enzyme while a value smaller than one signify a subsenstivity behaviour. This system was executed in GridCell and the steady state response from various quantity of enzyme $E1$ input is shown in Figure 3–12.

A Hill coefficient of 4 is observed in GridCell which is similar to the behaviour observed experimentally in Xenopus extracts and agrees to the numerical research conducted in [23]. The ultrasensitivity observed in the 3-stage MAPK cascade turns a graded input into a zero to one switch-like response. This kind of behaviour is believed to be necessary for signaling pathways driving biological process such as mitogenesis, cell fate induction and oocyte maturation, where the cell has to go from one discrete state to an other discrete state without intermediate grading.

CHAPTER 4
Architecture for a Stochastic Particle-Based Biological Simulator

Section 3.4 showed that the performance of GridCell executed serially on a generic CPU limits the simulation to systems both smaller than 10^7 voxels and converging in less than 10^6 to 10^8 time steps. Larger systems would not be able to produce results within one day of processing. In order to simulate larger and more complex systems, the algorithm needs to be expanded and accelerated. Fortunately, the algorithm has been designed with the idea that it would be executed in parallel right from the start of its conception. In this section, an FPGA architecture accelerating GridCell is presented. The architecture is described in VHDL and successfully synthesised on a Xilinx Virtex-6 device providing speedups of one to two orders of magnitude over the serial implementation.

The algorithm is to be executed on a software/hardware platform where a software CPU hosts the application and streams the data to be processed to an FPGA. All the initialisation, boundary and synchronisation issues are solved by the software processor, as in the serial implementation. However, the bulk of the simulation space is processed by the FPGA. The idea of splitting the work between the FPGA and the software processor has already been explored in applications such as traffic simulation [53] and produced large performance increases. The full simulation space is stored in the main memory of the system, while the local information needed by the FPGA is stored in the smaller and closer distributed memory on the FPGA. The data is streamed to the FPGA, which acts as a coprocessor. This approach ensures a fast execution while addressing the scalability issues of a direct implementation when simulating large systems. One big advantage of the CA model is that each individual voxel can be very simple and only takes a very little amount of memory. It is expected that two bytes of memory per voxel is sufficient to simulate moderately complex biological system. Therefore, with 4GB of RAM, a system containing $2 * 10^9$ voxels could be entirely contained in the main RAM system of the host processor. Larger systems could even be stored on a hard disk. The increase of processing power from the FPGA architecture makes it possible to increase the number of time steps by one to two orders of magnitude, which can provide either a better time resolution, a longer simulation or simply faster execution times.

In a parallel and streaming architecture such as GridCell, the bottleneck can either be the high bandwidth required to supply the FPGA or the amount of logic required on the FPGA. A bandwidth and memory analysis versus the size occupied in the FPGA is also conducted below.

4.1 6-Stage Pipeline

As mentioned in Chapter 3, the update process of the algorithm is divided into two steps; the movement phase and the reaction phase. As long as it is ensured that every particle passes through these two phases every time step, the two phases can be pipelined one after the other. The other condition is that they must work on independent voxels. Two voxels are independent as long as they don't share any of their immediate neighbourhoods. Therefore, each update process needs to have its own set of voxels along with their corresponding neighbourhoods which are not used by the other update process. In order to meet this goal, buffer layers are inserted between the movement phase and the reaction phase. In total, six different stages are needed, as shown in Figure 4–1. Stages 2 and 5 are the update and movement phase, while stages 1, 3, 5 and 6 are the buffer planes. Stages 1 and 3 are only accessed by the reaction unit, while stages 4 and 6 are only accessed by the movement unit. All stages consist of a 2D array of voxels. The size of these is determined by the number of processing engines (PEs) on each plane, which is in turn determined by the size of the FPGA and/or the bandwidth of the memory system. Once the movement and reaction phases have updated all the particles on their plane, all voxels are pushed to the next stage of the pipeline and the update process can begin anew. New data streamed from the host processor is written in stage one and the processed data of stage six is sent to the host.

The reaction phase is executed first to make sure every voxel attempts to react and move at every time step. If the movement phase was handled first, it could be possible for a particle to jump into a zone that has already been processed in a previous iteration, which would then prevent the particle from attempting to react. The iteration process of the pipeline is described in more details in Section 4.13.

Even though the algorithm uses a grid of type D3Q27, each voxel is only physically connected to either two or six adjacent voxels, depending on which stage they belong. As shown on Figure 4–2, all voxels on the four buffer stages are connected to their top and bottom neighbours only. The voxels on the two processing stages follow the arrangement shown in Figure 4–3 and have a direct access to the north, south, east, west, top and bottom directions. The processing voxels also interact with their corresponding PE, which is not shown on the diagram. Accesses to voxels outside these six cubic directions are made through a series of voxels. For example, a request to

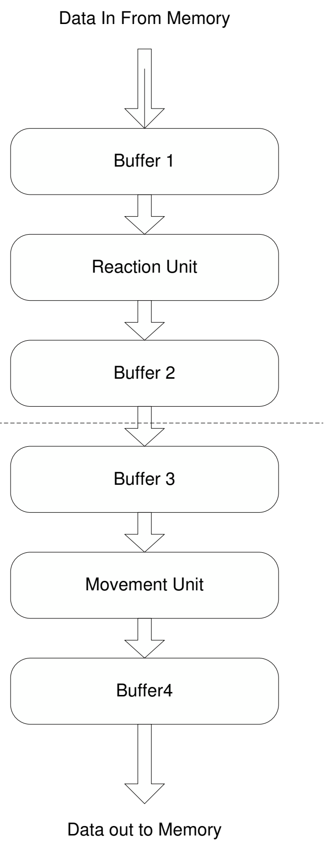

Figure 4–1: Side view of the 6-stage engine.

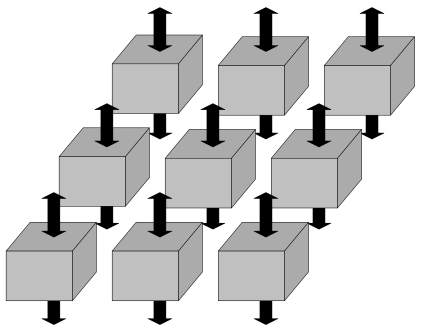

Figure 4–2: Voxel arrangement of the input and output ports on a buffer plane.

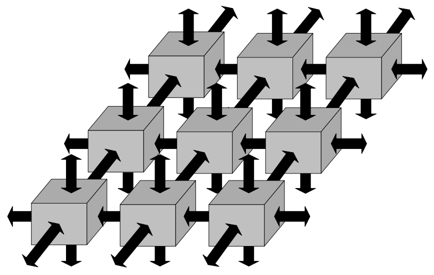

Figure 4–3: Voxel arrangement of the input and output ports on a processing plane.

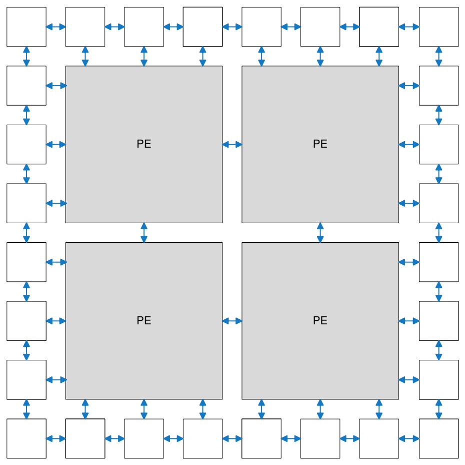

Figure 4–4: Arrangement of a 2x2 movement processing plane containing 4 MPEs.

the top northwestern voxel first goes through the north direction, then west direction and then top direction. This arrangement ensures the complexity of the interconnections between the voxels is kept to a reasonable level, at the cost of longer access time to farther elements.

4.2 Movement Plane

The movement plane is the fifth stage of the 6-stage pipeline. It contains interconnected movement processing engines (MPEs) in addition to an external layer of buffering voxels. The buffering voxels serve a similar purpose as the buffer planes. That is, in order to be able to update a voxel, its entire neighbourhood is required. This extra layer of voxels is the required neighbourhood needed to update all the voxels inside the MPEs. These buffering voxels are not actually processed, and they will still need to be updated by the processing elements in a following iteration. More details on the data flow of the update process is presented in Section 4.13. Figure 4–4 presents a plane consisting of four MPEs along with its layer of buffered voxels. In order to reduce the wasted bandwidth on buffered voxels, the shape of the MPEs should always be as close as possible to a square. For example, a 2x2 plane, a 5x5 plane, etc.

4.3 Movement Processing Engine

Movement processing engines are the main components of the movement processing plane. Each MPE contains 9 voxels interconnected into a small 3x3 square array, a random number generator, a finite state machine and the input/output ports required to connect to nearby MPEs and voxels in the buffer planes. Figure 4–5 illustrates a simplified architecture of the movement processing engine.

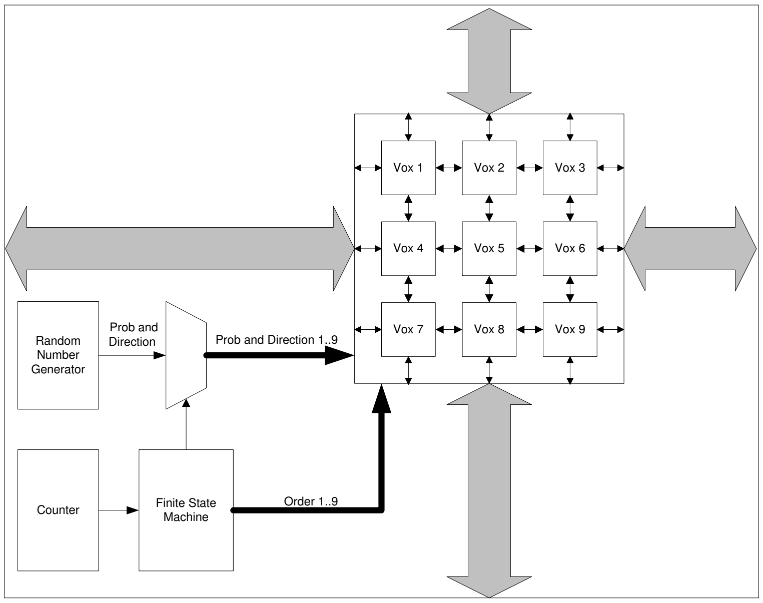

Figure 4–5: Architecture of the movement processing engine

Each MPE updates its nine internal voxels serially since their neighbourhood intersects with each other, causing a dependency between the voxels. However, as long as each MPE works on the same relative internal voxel, all MPEs can work in parallel. Figure 4–6 demonstrates the update process and shows how the neighbourhoods between voxels updated at the same time never intersect. An internal counter determines which voxel is currently being processed, and since all counters are synchronised, each MPE works on the same internal voxel.

The state of the FSM is driven by the value of the internal counter. The task of the FSM is to direct the data generated by the random number generator and issue a move command to the right voxel at the right time. It is also responsible to issue the push command to all voxels when all voxels have been updated. The push order writes the current voxel data into the next buffer plane while loading the data from the previous buffer plane. All other planes also perform the same push order at the same time, which ensures that all data move forward one stage at the same time and that nothing is lost or overwritten.

The movement phase of a single voxel can be performed in at most six clock cycles, as described in Section 4.4. Three extra clock cycles are required to push the data to the next stage and to accept the new data during the push order. Therefore, the movement phase takes 57 clock cycles to complete and the number of processed voxels during that time is equal to nine times the number of MPEs. Note that the reaction phase is more complex than the movement phase and takes more clock cycles to complete, which creates some free unused clock cycles during the movement phase.

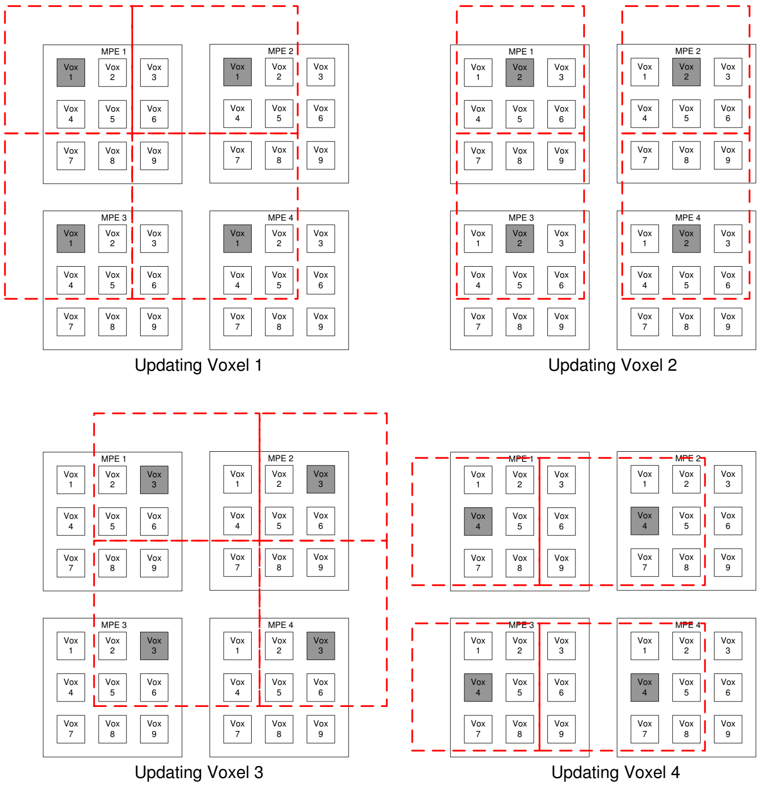

Figure 4–6: Update process of the MPEs. The red squares represent the neighbourhood of the voxels currently processed. By processing the same internal voxels, it is ensured the neighbourhoods of the updated voxels do not intersect.

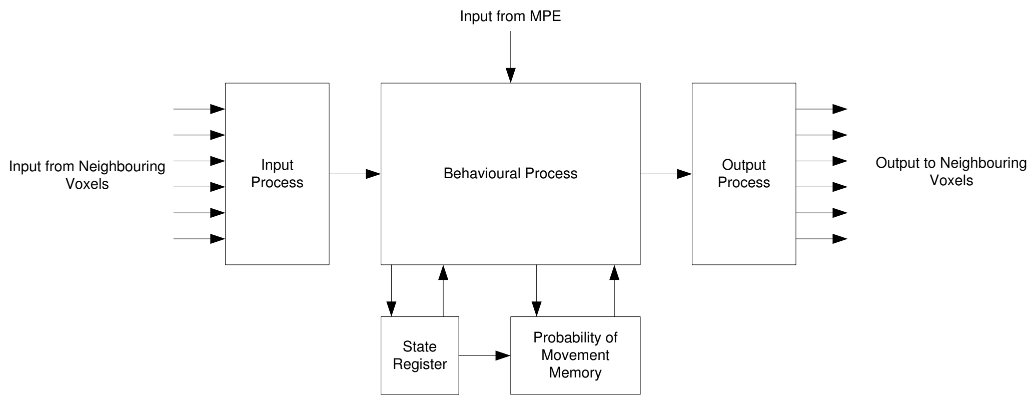
Input from MPE
Input from Neighbouring Voxels
Input Process
Behavioural Process
Output Process
Output to Neighbouring Voxels
State Register
Probability of Movement Memory

Figure 4–7: High level organisation of the movement voxel.

4.4 Movement Voxel

Voxels are the core components of the architecture. There are four different voxel versions in the architecture: the movement voxel located in the movement plane, the movement buffer voxel located in the buffer planes 4 and 6, the reaction voxel located inside the reaction plane and reaction buffer voxel located in the buffer planes 1 and 3. All share the same basic architecture with a few key differences. This section describes the movement voxel. Differences between the movement voxel and the other variations are described in their respective section.

Each voxel is designed as a CA entity, which implies its next state is calculated from its current state and from the state of its neighbourhood. For the purpose of the update process, the state of the MPE in which the voxel resides is also considered as being a neighbour. The voxel consists of a small register containing the state of the current voxel, the probability of movement memory block and three different processes: the input process, the behavioural process and the output process. Figure 4–7 demonstrates the block level architecture of the movement voxel.

4.4.1 Ports Description

Table 4–1 lists the input and output ports of the movement voxel along with their default size and a short description.

The task ports are 3-bit vectors, although only the two least significant bits are used in the moving part of the engine. Table 4–2 shows the list of possible tasks issued by both the MPE and the adjacent voxels.

The direction field is a 6-bit vector where each bit designates one of the six cubic directions. The six directions are north, south, east, west, top and bottom (N,S,E,W,T,B), and the direction field follows this specific order. A “1” value indicates the task needs to be forwarded toward the

Table 4–1: Input/output ports of the movement voxel.

Port	In/Out	Size (bits)	Definition
General Ports			
Clock	In	1	General Synchronous Clock
Reset	In	1	Synchronous Reset
Ports connecting to the MPE			
Task	In	3	Task to be performed by the voxel
Direction	In	6	Direction relative to the voxel where the task is issued
Prob	In	12	Random number generated by the MPE to compare to the probability of movement.
Ports connecting to adjacent voxels			Each of the following ports exist for the 6 cubic directions (N, S, E, W, T, B)
Species	In and Out	8	Species type of the moving particle
Compartment	In and Out	3	Compartment type of the voxel initiating the move task
Reacted	In and Out	1	Reacted state of the particle
Moved	In and Out	1	Moved state of the particle
Lifetime	In and Out	2	Lifetime remaining for temporary particle
Task	In and Out	3	Task to be performed by the voxel
Direction	In and Out	6	Direction relative to the voxel where the task is issued
Prob	In and Out	12	Random number generated by the MPE to determine if the move task is successful or not.

Table 4–2: Description of the movement task field.

Bitwise Value	Corresponding Task
000	Do nothing
001	Try to move toward given direction
010	Move is successful
011	Push the pipeline

Table 4–3: Fields of a voxel.

Field	Size (bits)	Definition
Species	4-12	Support for 16 to 4096 different species
Compartment	3-4	Support for 8 to 16 different compartments
Reacted	1	Specify if the particle has already reacted on the current time step
Moved	1	Specify if the particle has already moved on the current time step
Lifetime	2	Lifetime of a temporary species before reverting back to its previous state

given direction. For example "100110" would designate the north west top direction. Although it is not used in the movement phase, nothing prevents two opposite directions on the same axis to be active at the same time. This feature is used in the reaction portion of the architecture when multiple concurrent accesses are issued.

The species, compartment, reacted, moved and lifetime ports are used to send and receive the information regarding the state of the moving particle. More details on their specific meaning is provided in the next section.

4.4.2 State Register

This register contains the information necessary to describe the state of a voxel. Table 4–3 describes the various fields. The size of most entries can be modified depending on the size of the systems expected to be simulated. Most biological systems should require between 11 and 20 bits to completely describe a voxel. The default and current size of the hardware implementation is 16 bits. It is large enough to support a number of species and compartments for moderately complex systems while making memory alignment as simple as possible and keeping the memory requirements low. With 16 bits, the simulator can support 512 different species and 8 different compartments. If memory space or bandwidth is a problem, we have the flexibility of using smaller voxels and, for very large systems, it is also possible to increase voxel size.

4.4.3 Probability of Movement Memory Block

As described in Chapter 3, the probability of movement at each time step is calculated from the diffusion speed of each species in each different compartment. The number of entries is equal to the supported number of species multiplied by the number of compartments squared. Using the default values, $512 * 8 * 8 = 32768$ different entries are needed. Each entry contains a 12-bit number representing the probability of movement per time step. Thus, the total amount of memory needed per voxel is 393.2 Kbits. Considering even the current largest Virtex-6 device contains a maximum 34 Mbits of internal memory (BRAMs and distributed memory included), the maximum number of voxel supported is 86, and this is without considering any other memory elements needed in other parts of the design. This is clearly below the 588 movement voxels contained in a 16 MPEs architecture, an implementation size believed to provide a speedup larger than one order of magnitude.

As the brute force approach is not conceivable, a more flexible storage structure is explored. By assuming the moving matrix is sparse, that most species have a variable number of entries and that most simulation spaces do not use the maximum number of supported species, it is possible to dramatically reduce the memory requirements of the probability of movement matrix. The new memory structure uses a two layer architecture where all entries contain useful non-zero information. The first layer consists of a memory block indexed by the species number containing the address of the first element and the number of elements belonging to that species in the second block. The second block contains all the probability of movement data sorted by species number as well as the values of the two compartments corresponding to each entry. In addition to the maximum number of entries contained in the second layer memory block, a restriction on the maximum number of entries per species is enforced. This second limitation is necessary in order to maintain a constant and fast execution speed. Indeed, by limiting the number of entries per species to a specific value, we can design the second memory element in such a way that it always provides the maximum number of entries in a single read. A distributed memory structure made of several smaller memory banks with interleaved data is designed. Assuming N is the maximum number of entries per species, the memory block also contains N different memory banks. By multiplexing the inputs and outputs, a single read accesses all N small memory blocks and provides the data at the requested address as well as the next $N-1$ entries. The idea of distributed memory with interleaved data has been explored in [55]. Figure 4–8 shows the general memory structure. Figure 4–9 shows an example of the architecture of the second layer memory element with four different memory banks. The current hardware implementation uses eight banks generated through the Xilinx IP core distributed RAM structure. Finally, once the data from the distributed memory block is obtained, a selector block compares the value of the two compartment values from the distributed memory block to the two compartment values involved in the movement task. If a match is found, the corresponding probability of movement is sent. If no match exists, it means the current combination of compartment is not valid and the probability of movement is zero.

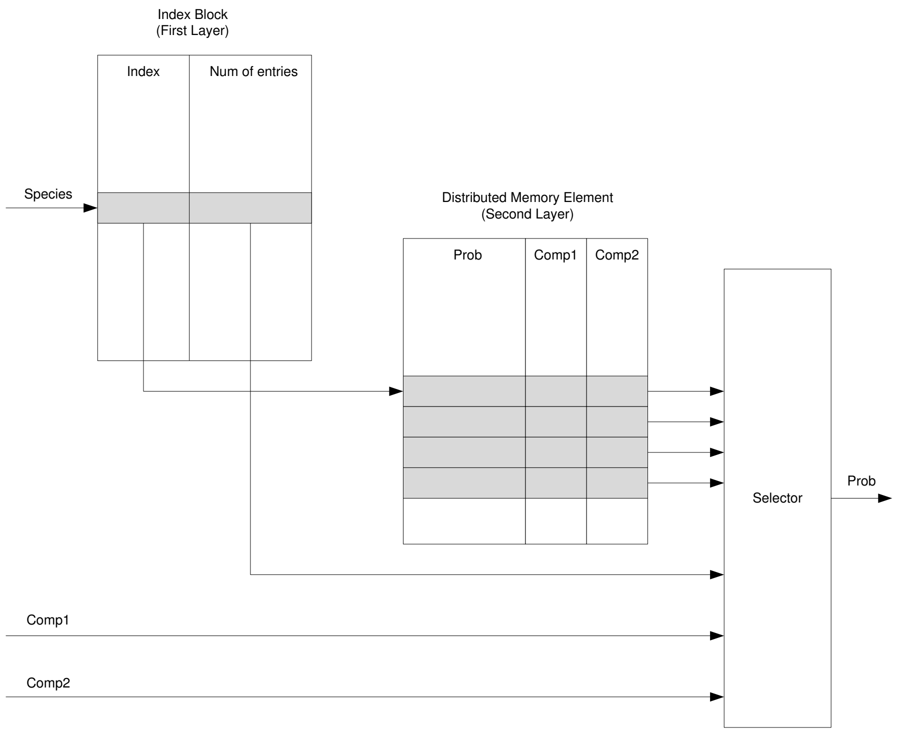

Figure 4–8: Architecture of the probability of movement memory block. First block receives the species type and output the address of the first element and the number of elements N belonging to that species to the second block. The second block reads the address and output the next N following entries. The selector compares the compartment values and if a match is found, outputs the corresponding probability of movement value

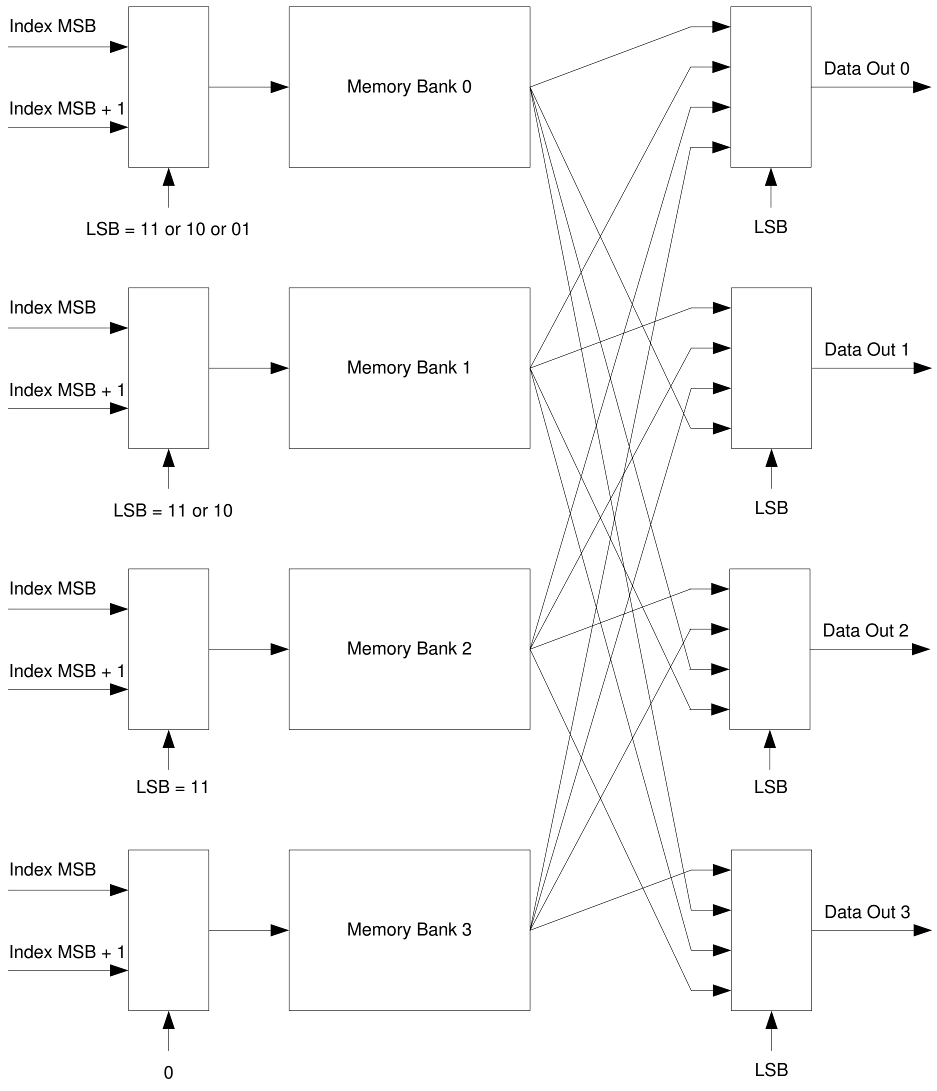

Figure 4–9: Structural view of the distributed interleaved memory block with four distinct memory banks. A front layer of multiplexors selects the right index for each memory bank based upon the values of the least significant bits of the address. A back-end layer of multiplexors reorders the data from the memory banks to the output.

The maximum number of entries per species has a minimal impact on the total amount of memory required for the total structure, as only one extra bit per species is added every time the maximum value is doubled. The flexible structure of FPGA distributed memory also makes it possible to modify this parameter without modifying the general architecture of the memory blocks if the modelling accuracy is affected by this limitation.

The main parameter affecting total memory space is the total number of entries supported in the second block. The maximum number of entries is currently set to 512, a number believed to be large enough for most current models. By setting the maximum number of entries to 512 and the maximum number of entries per species to eight, the total amount of memory required is 16Kbits, which is 25 times smaller than the brute force approach. As mentionned earlier, both the total number of entries and the number of entries per species can be increased without modifying the architecture if the model requires more species and more compartments. While the maximum number of entries per species has a minimal impact on the memory and logic cost, increasing the total number of entries would increase the required amount of memory per voxel which would impact the maximum number of voxels that can be fitted on the FPGA device.

Another alternative can be explored to reduce the memory requirements of the probability of movement data. It involves simplifying the algorithm so that each species has a single diffusion speed per compartment and that any transition between compartments would use the diffusion value of the destination compartment. For example, a particle going from compartment A to compartment B would simply use the diffusion speed of compartment B instead of using the probability of movement of going from compartment A to compartment B. The moving ratio memory element would then scale linearly with the number of compartments. However, implementing this modification would remove the capacity to have different diffusion speed when entering and exiting a compartment, a feature useful to simulate active transport.

4.4.4 Input Process

The input process reads all the inputs from the six neighbouring voxels and generates a single signal used in the behavioural process. Over the multiple iterations of the design of the FPGA implementation, this process went from a fairly complex, priority-driven multiplexor to a much lighter and faster block performing a 6 to 1 “or” operation to all voxel inputs. The architecture is made in such a way that if more than one input is active on the same clock cycle from multiple directions, then all inputs are necessarily the same request coming from different paths. Therefore, the “or” logic is a small and fast operation making sure that all ports are listened to and that the format of the data is preserved.

The input process also updates the value of the direction port in order to reflect the new relative position of the voxel. Any input coming from a given direction is now necessarily at the right location on the given axis. Therefore, the direction bits of that axis are set to ‘0’. For example, a voxel receiving a task from the south port with the direction “100110”, which designates the north west top direction, updates the direction vector to “000110” by setting the NS direction bits to 0, as the task has travelled north and is now at the right location on the NS axis. The move request now only needs to go to the west and top direction.

4.4.5 Behavioural Process

The behavioural process is a finite state machine reading the signal sent by the input process and the MPE to produce the new state of the voxel. The process first looks at the MPE ports. The MPE can request two different tasks. The first one is the movement task, to which the voxel responds by forwarding the task with its current voxel information to the direction and with the probability number provided by the MPE. The second task is the push task, where the voxel forwards its voxel information to the voxel in the bottom direction. The push order starts a cascade of push tasks in the other planes, so that all the voxels in a given plane are forwarded to the next plane. If no task is requested by the MPE, the process looks at the signal generated by the input process coming from the neighbouring voxels.

If the task is a movement order, the behavioural process first looks at the direction field. If it is not equal to “000000”, it means that the task has not yet arrived at destination. The process then forwards the received task to the correct direction(s) and updates an internal direction register which remembers the direction from which the order came from. If the direction is equal to “000000”, the task has arrived at destination. The process then checks if the voxel is empty and if the received probability is smaller than the probability of movement. If both checks succeed, the movement is successful and the voxel updates the *species*, *reacted*, *moved* and $lifetime$ fields with the data provided by the input signal. The process then sends a reply to the direction(s) from which the movement task came from, mentioning the movement is successful.

If the process receives a movement successful order, it checks its internal direction register. If it is not “000000”, it means that the voxel previously forwarded a move order and it implies that we are not at the right location yet. The process then forwards the task to the direction indicated by

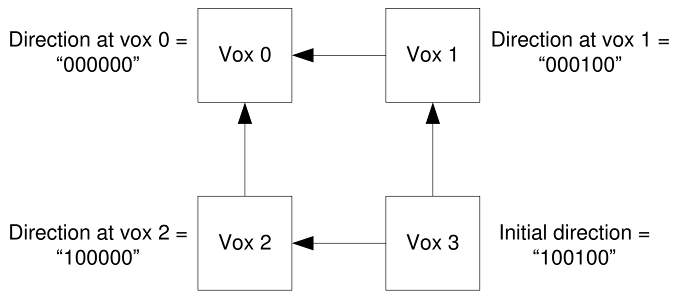
Direction at vox 0 = "000000"
Vox 0
Vox 1
Direction at vox 1 = "000100"
Direction at vox 2 = "100000"
Vox 2
Vox 3
Initial direction = "100100"

Figure 4–10: Example showing voxel 3 performing a movement order to the NW direction. The voxel forwards the task to voxel 2 to the west and to voxel 1 to the north. In the next clock cycle, voxel 1 and voxel 2 both forward the task to voxel 0 to the west and north respectively.

the internal direction register. If the register is all zero, then we are back at the original location, and the voxel is emptied.

Finally, if the process receives a push task from a top voxel, the process overwrites its current voxel registers with the data provided by the top voxel. As the request came from the top voxel, it is assumed that the current voxel information has already been forwarded to the next stage a few clock cycles before when the MPE requested the initial push order.

4.4.6 Output Process

The output process is in charge of reading the direction field generated by the input and behavioural process and drives the output ports. For every active bit in the direction field, the process sends the data received from the behavioural process to the corresponding direction. For every inactive direction, zeros are outputted. It is not unusual that multiple ports are active at the same time and, while it is not a necessity in the movement part of the pipeline, it does not influence the end result. Figure 4–10 shows an example in which voxel 3 attempts to move to the NW direction. The voxel forwards the task to voxel 2 to the west and voxel 1 to the north. In the next clock cycle, voxel 1 and voxel 2 both forward the task to voxel 0 to the west and north directions respectively. Assuming a successful movement, voxel 0 will reply in a similar fashion in the opposite direction.

4.4.7 Pipelining of the Movement Phase

As the farthest possible neighbour a voxel can interact with is three steps away, it takes three cycles to reach the neighbour and another three cycles for the reply to reach the initial voxel. This implies a maximum of six clock cycles is needed to perform the movement task of a single voxel. However, as it will be shown in the reaction voxel section, the reaction phase needs three extra clock cycles to perform the update, requiring up to nine clock cycles per voxel to complete. As both update processes need to be synchronised, this leaves an opening of three clock cycles where no actual work is done. These three clock cycles provide some room to pipeline the slowest stages of the movement update process and increase the throughput of the movement engines. The critical data path starts at the *species* register of the active voxel, needed by the probability of movement memory block, which drives the selector before being compared to the probability value. If the movement is successful, it then triggers a "move successful" reply. By adding a register between the first and second memory blocks, between the second memory block and the selector and between the selector and the update process, the overall update process now takes three additional clock cycles and a speedup of 2.2x is achieved.

4.5 Random Number Generation with LFSRs

Each MPE needs a random number generator (RNG) to provide the *direction* and the *probability* values. The direction needs to be random enough to simulate the Brownian motion of the particle and to make sure that no artificial internal flow is imposed on the particles. On the other hand, the probability value needs to be uniform enough to avoid modifying the overall rate of movement of the particle.

Generating true random numbers on FPGA is impractical; the usual method is to generate pseudo-random numbers with the help of linear feedback shift registers (LFSRs) which can be efficiently mapped in modern FPGAs. The random number generator uses a combination of 10 16-bit LFSRs to create a 32-bit pseudo-random number. Out of these 32 bits, 12 are used as the uniform probability value and the remaining 20 bits are multiplied by 26 to create a uniform integer number ranging between 0 and 25 inclusively. This number is mapped through a LUT to a unique direction. The multiplication is implemented as a series of shifted additions since one of the two inputs is fixed.

While it is difficult to determine the quality of randomness of a random generator, the National Institute of Standards and Technology (NIST) [48][41] developed a battery of statistical tests to detect non-randomness in binary sequences generated by random and pseudo-random number generators. The primary test is called the frequency monobits test and is essentially comparing the number of '0' values to the number of '1' bits in a string of random number. Ideally, both values should be generated very close to 50% of the time. The frequency monobits test has been applied to GridCell's RNG for various seeds and length sequences and it succeeded all the tests. Additional testing on the software version of GridCell has shown that switching from a reliable double-precision random number generation library to a bit-accurate C version of the FPGA RNG did not introduce any observable change in the results of the systems demonstrated in Section 3.

Given that GridCell's algorithm needs uniform random values at irregular intervals, it is not really sensitive to slight patterns that might be observed from using an imperfect RNG. Also, contrary to most cryptographic applications, the RNG does not need to generate values which are theoretically impossible to predict based on the past generated values and in general, does not need to be as strong. Therefore, while not being a complete analysis of the RNG used in the FPGA, the frequency monobits test and the practical tests validate that the quality of the RNG is high enough to not introduce any significant bias in the system.

4.6 Movement Buffer Plane

The movement buffer plane consists of a two dimensional array of movement buffer voxel matching the size of the movement plane. Two different movement buffer planes are present in the 6-stage pipeline, one in stage 4 and one in stage 6.

4.6.1 Movement Buffer Voxel

The movement buffer voxel is a stripped down version of the movement voxel. First, since it is not located inside an MPE, it does not need any port and logic dealing with tasks coming from a MPE. It also only has connections to the top and bottom voxels which reduces considerably the complexity of the wiring of the movement buffer plane, as all the NSEW movements are handled in the main movement plane. The input and output processes behave in the same way as the movement voxel. The behavioural process behaves similarly to the movement voxel except when a push task is issued from the top voxel. In that case, instead of only overwriting the state of the voxel with the data coming from the top voxel, it also sends its current state to the bottom voxel. This ensures that the cascade of push commands is forwarded to the next stage. Only a non-buffer movement or reaction voxel can start or stop a push cascade.

4.7 Reaction Plane

The reaction plane is the second stage of the 6-stage pipeline. The reaction plane is similar in structure to the movement plane and the organisation of Figure 4–5 is applicable to the reaction plane as well. The two main differences are the use of reaction processing engines (RPEs) instead of MPEs and the external layer of voxels is made of the reaction voxel type.

4.8 Reaction Processing Engine

From a structural point of view, the RPEs are similar to the MPEs. Each RPE contains nine reaction voxels organised in a 3x3 array and the order of processing is done in a similar fashion.

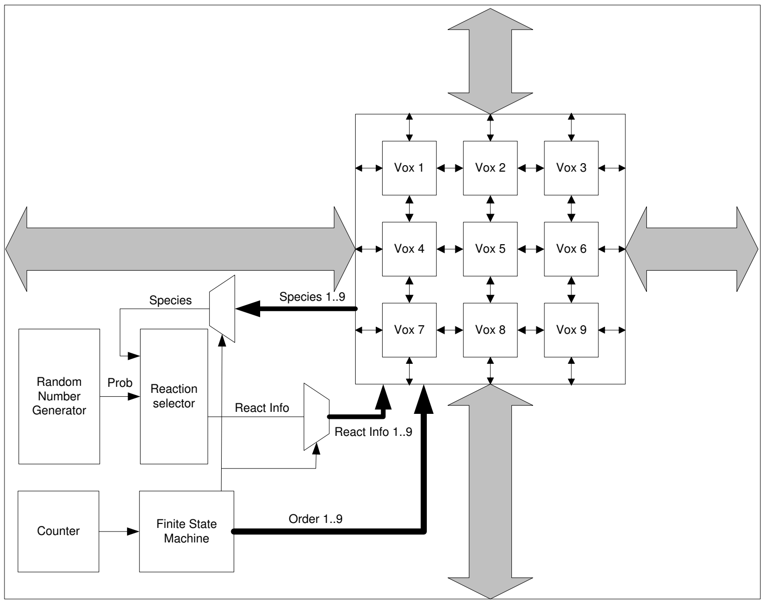

Figure 4–11: Architecture of the reaction processing engine

Each RPE also contains the same RNG although RPEs are only using the *prob* output. The FSM is similar in purpose in that it acts as a big multiplexor connecting the right data and tasks to the right voxel at the right time and is driven by an internal counter. One of the main differences is the presence of the reaction selector block which selects which reaction the active voxel attempts to perform. Figure 4–11 shows the architecture of the RPE.

4.9 Reaction Selector

The reaction selector takes into consideration the species number of the particle attempting to react and the random probability number from the RNG to generate the reaction information. The reaction information includes the type of reaction which specifies if a merge, transform or split reaction is requested and the species of the other particle involved in the reaction, if applicable. The reaction selector is a memory structure very similar to the probability of movement memory block shown in Figure 4–8. It is a dual layer architecture where the first memory block is indexed by the species number and outputs the address of the first entry as well as the total number of entries of the second block. The second block is a list of all the reactions with their probability of reaction in order of species number. The second block is also a distributed memory consisting of several smaller memory banks with interleaved data similar to the block shown in Figure 4–9. It always outputs the maximum number of reactions per species supported by the implementation. The memory banks are generated by using the Xilinx IP core software to create fixed sized distributed RAMs. Finally, a selector takes as input the maximum number of reactions from the first block, the random probability from the RNG and the reaction data from the second block to select which reaction is selected.

The current implementation supports a maximum of 512 different reactions with a maximum of eight different reactions per species. These two numbers are large enough to support moderately complex systems and as in the case of the movement phase, the maximum number of reaction per species can be increased for a minimal increase in memory fabric while increasing the maximal number of reaction will require more memory per RPE, which could reduce the number of PEs that can be stored on the device. Note that contrary to the movement memory block, which is currently needed in every movement voxel, only a single reaction selector per RPE is required, which reduces considerably the memory scaling issues of increasing those two parameters.

As was the case with the movement, the data path starting at the *species* register of the active voxel going into the two layers of memory and the selector of reaction to the active voxel to start the reaction update process is by far the longest path in the architecture. In the movement voxel, we had three free clock cycles that were not used since we had to wait for the reaction update process anyway. In this case, the three clock cycles are needed to finish the reaction update phase. Fortunately, the properties of the architecture allow us to prefetch the species number information of the next active voxel before the current reaction update process is finished. Indeed, since all the reaction voxels inside the RPEs are on the same plane, they are at most two steps away from any other active reaction voxels, meaning that any voxel on this plane is updated within six clock cycles at the most. Only the voxels in the buffer planes can take up to nine clock cycles to update. As such, we can be certain that by clock cycle 6, the species information of the next active voxel is the correct one. It is then possible to start the reaction selector process at that time and pipeline the process as we did to the movement phase so that the selection of the reaction is already done when it is time to start updating the next particle. Doing so provided a speed up of 2.2x compared to the version where no pipelining and no prefetching was done. The speed up is in the same order as the one observed for the movement voxel and both the reaction and the movement phases run at about the same speed.

4.10 Reaction Voxel

The reaction voxel is structurally similar to the movement voxel shown in Figure 4–7. The main difference is that it does not need any probability of movement memory structure. The reaction selector, which serves a similar purpose in the reaction update, is located in the RPE and is shared for all reaction voxels. The behavioural process is also modified to handle reaction tasks.

Table 4–4: Input/output ports of the reaction voxel.

Port	In/Out	Size (bits)	Definition
General Ports			
Clock	In	1	General Synchronous Clock
Reset	In	1	Synchronous Reset
Ports connecting to the MPE			
Task	In	3	Task to be performed by the voxel
species1	In	8	First species type involved in the reaction
species2	In	8	Second species type involved in the reaction
speciesout	Out	8	Species type of the current voxel. This is used by the reaction selector
Ports Connecting to buffer voxels			Each of the following ports exist for the 2 directions (T, B)
Species	In and Out	8	Species type of the moving particle
Compartment	In and Out	3	Compartment type of the voxel initiating the task
Reacted	In and Out	1	Reacted state of the particle
Moved	In and Out	1	Moved state of the particle
Lifetime	In and Out	2	Lifetime remaining for temporary species
Task	In and Out	3	Task to be performed by the voxel
Direction	In and Out	6	Direction relative to the voxel where the task is issued
Prob	In and Out	12	Not used.
Ports Connecting to reaction voxels			Each of the following ports exist for the 4 directions (N, S, E, W)
Species	In and Out	8	Species type of the moving particle
Compartment	In and Out	3	Compartment type of the voxel initiating the task
Task	In and Out	3	Task to be performed by the voxel
Direction	In and Out	6	Direction relative to the voxel where the task is issued

4.10.1 Ports Description

The ports of the reaction voxels are slightly different from those in the movement voxel. Table 4–4 shows the input and output ports, the default number of bits, as well as a short description of each port.

The $lifetime$, *prob*, *reacted* and *moved* data fields are not needed to perform a reaction, which is why the buses on the reaction plane are smaller. The top and bottom ports keep the same format as the movement voxel since it is through these ports that the push order is issued. The push command can also be considered as a forced move command. Probability is not used but is kept as a port in order to keep some uniformity in the voxels; otherwise, six different variants, one for each stage, would be required instead of four.

The task field in the reaction part of the pipeline has different meanings. Table 4–5 describes the possible values and their corresponding meaning.

4.10.2 Input Process

The input process of the reaction voxel uses the same "or" logic as the one from the movement voxel for all inputs except for the internal register storing the direction from which the input(s)

Table 4–5: Description of the reaction task field.

Bitwise Value	Corresponding Task
000	Do nothing
001	Perform a merge reaction
010	Unused
011	Push the pipeline
100	Perform a transform reaction
101	Perform a split reaction
110	Reply from a neighbouring voxel to the active voxel saying that a match has been found
111	Reply from the active voxel to a matching voxel to complete the reaction

came from. The movement voxel was simply "or"ing all the directions together, which meant that the voxel could reply to multiple directions at once. We can do so since the movement task is targeting a single direction, so the path itself is not important as long as the origin and destination remain the same. In the case of the reaction voxel, for both the split and the merge reaction, a search is conducted in the full neighbourhood in order to find the required match. Therefore, all voxels are targeted and tested, and out of all the possible matches, only a single one can be selected. A priority system is needed to make sure only a single external voxel is affected by a single reaction. Thus, if a voxel receives the same task from two different directions and needs to reply, the reply is sent to the direction with the highest priority.

In order to eliminate any bias on the macro-level caused by keeping a static priority order, the priority order is toggled between two opposite directions every time steps during the push order. The order, from most important to least important, switches from 1) north, south, east, west, top, bottom to 2) south, north, west, east, bottom, top. These two opposite alternating directions prevent the creation of any artificial flow of particles that would be caused by always creating or deleting particles in the same direction.

4.10.3 Behavioural Process

The behavioural process is a finite state machine reacting to the signal sent by the input process and the RPE to produce the new state of the voxel. Similarly to the movement voxel, the process looks at the RPE first and if no task is requested, the inputs coming from adjacent voxels are considered. The RPE can request four different tasks. Three of the four tasks correspond to the three different types of reaction, the transform reaction (task = "100"), the merge reaction (task = "001") and the split reaction (task = "101"). The last task is the push order (task = "011"), which is handled in the exact same way as in the movement voxel.

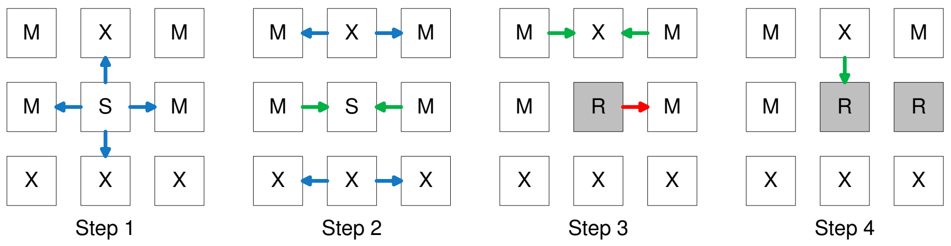

Figure 4–12: An example of a 2D broadcast search. Voxel S is searching for a specific species type in its neighbourhood; the matches are designated by the M notation, while Xs are mismatches. On step 1, S is sending a broadcast search to its four direct neighbours (blue arrow). On step 2, the west and east voxel, which are both matches, reply with a match found signal (green arrow). The north and south voxel, being mismatches, forward the broadcast search to the last layer of neighbours. On Step 3, S reacts, preventing any other reaction from happening during this time step, and sends a confirmation (red arrow) to the east voxel since it is higher on the priority list than the west one. The northwestern and northeastern voxels reply to the north voxel that a match has been found. On step 4, the east voxel reacts and the process is finished. S ignores the second wave of reply from the north since it has already reacted.

The transform reaction first verifies if the particle has not already reacted during this time step. If the particle can react, the species type of the active voxel is replaced with the new type provided by the reaction selector, and the state of the particle is set to *reacted*.

The merge and the split reaction are handled similarly. The *reacted* state is also verified and if it is zero, a broadcast search is sent to the full neighbourhood of the active voxel. The difference between the split and the merge reactions is that the merge reaction searches for a specific particle type, while the split reaction searches for any empty voxels. Figure 4–12 shows an example describing how the broadcast search is conducted. The advantages of this broadcast search are numerous. First, the execution time is independent of the number of good or bad matches. Second, if more than one match is found, only one of them is chosen and the selection is done automatically. Finally, if no match is found, no reply is sent and the reaction is aborted.

When no task is active from the RPE, the process listens to the adjacent voxels. If the task comes from a broadcast search from a split or a merge reaction, the process checks three conditions: 1) The compartment numbers must match as no reaction between different compartments can be triggered. 2) The particle must not have already reacted. This check is not relevant if the voxel is empty. 3) The particle types must match. If all these conditions are true, a reply is sent back toward the highest priority direction saying a match has been found (task = “110”). If one of the conditions is false, the broadcast is forwarded to the next layer of neighbouring voxels.

If the task indicates that a match has been found (task = “110”), the process tests the internal direction register and the *reacted* state to make sure that we are at the right location and that the particle has not yet reacted. If both tests pass, the voxel is updated with the new *species*, *reacted* is set to true and a reply (task = "111") is sent back toward the highest priority direction from which the response(s) was(were) received. This reply is necessary because even if the external particle is a match, it can not know that it is the one that is selected until the reply is received. When the internal direction register is not zero, the task is forwarded in the direction saved by the internal direction register.

If the task indicates a confirmation (task = "111") order, the process tests the internal direction register. If it is zero, we are at the right place, and the species voxel is replaced either with the corresponding species or is emptied. If the internal direction register is not zero, the task is forwarded.

Finally, the push task (task = "011") is handled the same way as the movement voxel.

4.10.4 Output Process

The output process is identical to the one in the movement voxel except for the format of the input/output ports, which is slightly different.

4.11 Reaction Buffer Plane

The react buffer plane is a two-dimensional array of reaction buffer voxels matching the size of the reaction and movement plane. A first reaction buffer plane is located in stage 1 the other one is located in stage 3.

4.12 Reaction Buffer Voxel

Similarly to the movement buffer voxel, the reaction buffer voxel is a stripped down version of the reaction voxel. All the ports and logic related to the RPE have been removed as well as all the input and output ports connecting to the NSEW directions. As was the case with the movement update, all the NSEW requests are handled in the main reaction plane. The input and output processes behave in the same way as the reaction voxel. The behavioural process behaves similarly to the reaction voxel except when a push task is issued from the top voxel. In this case, instead of only overwriting the state of the voxel with the data from the top voxel, it also sends its current state to the bottom voxel.

4.13 General Update Process

Assuming that the north-south dimension is considered to be on the y axis, the east-west dimension is on the x axis and the top-bottom dimension goes along the z axis, the pipeline processes 6 planes in the x and y axis and moves along the z axis after each xy plane is processed. In general, the size of the xy planes of the pipeline is smaller than the size of the full simulation

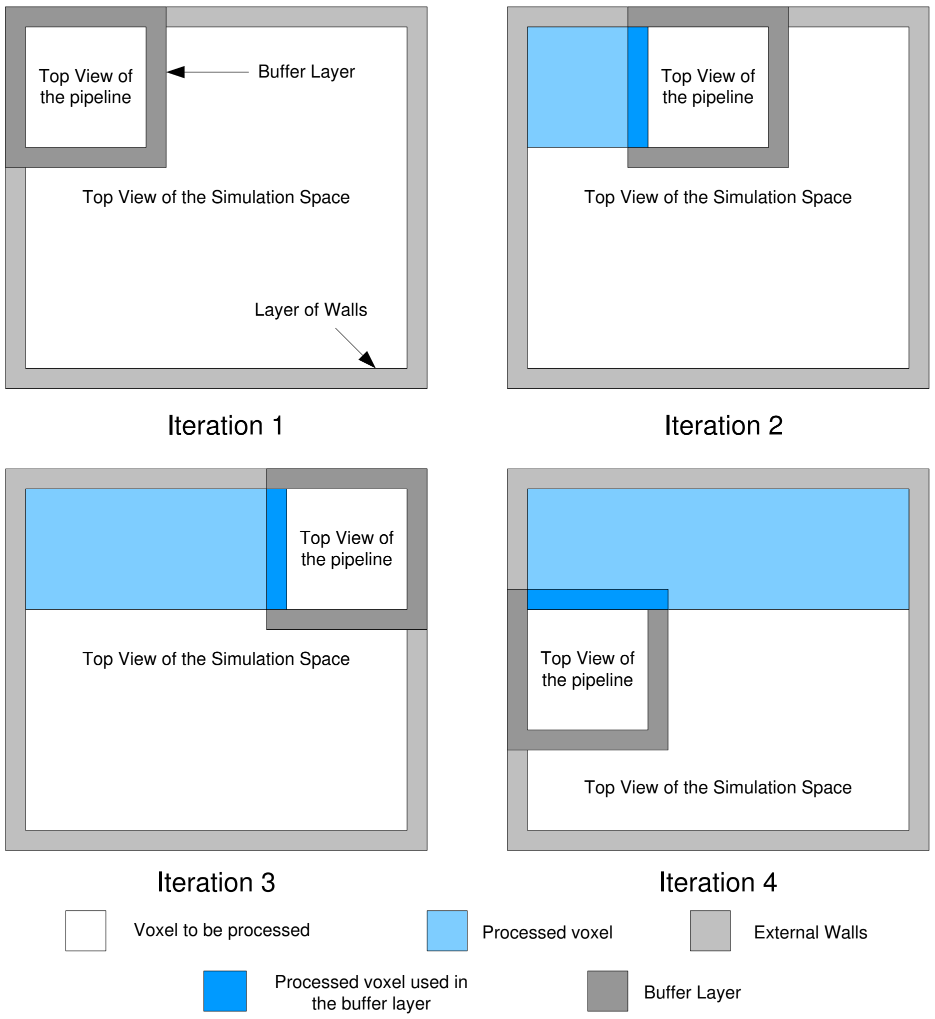

Figure 4–13: Pipeline movement over multiple iterations to update the full simulation space. After each iteration, more and more of the simulation space is processed. In this example, it would take roughly nine iterations to cover the full space. Top view of the xy plane is shown.

space. Therefore, it takes several displacements and iterations of the pipeline to cover the full simulation space. Figure 4–13 shows an example describing how the pipeline moves in order to cover the full simulation space.

The size of the simulation in the xy axis does not necessarily have to be an integer number of times the size of the pipeline. A smart memory controller could start processing the next row right away with the remaining processing elements as long as the two layers of walls delimiting the end of the current row and the start of the next one are included. This arrangement would ensure the processing elements would always be busy and doing effective work. A simpler controller could fill the rest of the processing elements after the wall with empty voxels and could start on the next row on the next iteration. The same principle can also be applied when finishing a time step and starting the next one.

The x, y, z planes are all interchangeable, as rotating the simulation space has no effect on the end result of the simulation. Optimally, in order to reduce the amount of memory management required when starting a new iteration, the number of iterations should be minimised. Therefore, the z axis should always be the longest axis of the simulation space.

4.14 FPGA Implementation

The architecture is synthesised using the Xilinx ISE 12.2 software and simulated with ModelSim XE 6.5. The FPGA target device is the latest generation Virtex 6 XC6VLX760 which is one of the two largest devices currently available from Xilinx. The goal was to validate the architecture used for the implementation of the GridCell algorithm and evaluate the range of speedups obtainable with current technology. As the architecture is scalable, smaller and (future) larger FPGA can be used with little to no modifications. The number of processing elements can be modified to fit the bandwidth as well as the size of the different devices and systems.

4.14.1 XC6VLX760 Xilinx Virtex 6 Device

The FPGA XC6VLX760 contains 118560 slices, each of them made of four 6-input LUT and eight flip-flops, which correspond to a rating of 758784 logic cells. Some of the slices can be used as distributed memory or shift registers, two functions extensively used in the architecture. The device also has 1440 18 Kb BRAMs containing up to 26 Mb of internal memory. BRAMs are not currently used in the design as they are rather large memory banks and GridCell makes a better use of multiple smaller memory elements provided by distributed memory. However, if distributed memory becomes limited, they would make a fine additional memory resource to share the memory load. The first target would be to replace the reaction memory blocks with BRAMs. Several hundreds of DSP slices, each containing a multiplier, an accumulator and an adder, are also featured in the device. However, the algorithm requires a very limited amount of complex mathematical calculations, and even the smallest FPGA has more than enough DSP units to cover several hundred PEs.

4.14.2 FPGA Synthesis Resources Utilisation

Four different versions of the architecture are synthesised. The first one contains a single MPE and RPE, the second one forms a 2x2 array of PEs (4 of each), the third version is made of a 4x4 array of PEs for a total of 16 MPEs and 16 RPEs and, finally, the fourth version consists of a 5x5 array. Table 4–6 shows the resource utilisation of each version of the pipeline.

Table 4–6: Resource utilisation of the XC6VLX760 FGPA.

Number of PEs per stage	Slice Registers (%)	Slice LUT (%)	Clock Speed (MHz)
1	7300 (0%)	17500 (3%)	182
4	29720 (3%)	67576 (14%)	180
16	117797 (12%)	275473 (58%)	177
25	184022 (19%)	460279 (97%)	173

Table 4–7: Throughput and gains of performance of the architecture.

Number of PEs per stage	Throughput (MVox/s)	Speedup
1	19	3.2
4	75	12.6
16	296	50
25	452	76

The number of resources used scales linearly with the number of processing elements which is expected. The clock cycle speed is also mostly unaffected by the size of the pipeline. The small slowdown in speed is explained by the fact that, while the critical data path does not change, as more and more units are added, the worst routing delay may increase slightly.

Looking at the trend from table 4–6, as we increase the number of processing elements, the number of Slice LUT is the resource saturating first, which is usually the case in most FPGA designs. With 25 PEs per stage, the device is almost completely used.

4.14.3 Performance

The throughput in voxels per second of the pipeline is given by

$$T_{vox} = \frac{9FreqN_{PE}}{86}, \tag{4.1}$$

where $Freq$ is the frequency of the FPGA pipeline and N_{PE} is the number of MPEs on the movement and RPEs on the reaction stage. The number 9 is the number of voxel per PE and the number 86 is the number of clock cycles required to fully update the PEs (9 voxels * 9 clock cycles + 3 for pushing the pipeline + 2 for pipelined latency delay of the first voxel).

Given a frequency of 172MHz and 25 MPEs, we obtain a throughput of 452 Mvoxels/sec. By calculating the throughput of the software version with the same settings as those used for table 3–1, we may observe that the FPGA implementation is over 76 times faster than the software version. Table 4–7 presents the speedups achieved with the different implementation sizes of the architecture. Each additional PE present in the pipeline roughly increases the gain of performance by an additional 3x over the serial implementation.

In order to calculate the memory bandwidth required by the architecture, we first need to compute the number of voxels present per plane, including the buffer zone. Assuming a square number of processing elements, the relationship between the number of voxels per plane N_{vox} and the number of PEs N_{PE} is given by

$$N_{vox} = (3\sqrt{N_{PE}} + 2)^2. \tag{4.2}$$

Then, the memory bandwidth required to supply the pipeline is given by

$$BW = \frac{N_{vox} V_s Freq}{86} \tag{4.3}$$

where BW is the bandwidth required by the movement stage, N_{vox} is the number of voxels per plane, $Freq$ is the clock frequency of the FPGA board and V_s is the number of bits per voxel.

Assuming a frequency of 172MHz, 25 MPEs and a voxel size of 2 bytes, we obtain a bandwidth of 1.2 GB/s per direction, for a combined 2.4 GB/s memory bandwidth. This range of bandwidth is within the range of most current systems. For example, the SGI RC-100 Numalink interface can support theoretical speeds of up to 6.4 GB/s.

4.14.4 Scaling Properties and Bottlenecks

From the bandwidth analysis and the resource utilisation, we can see the resource utilisation of the Slice LUT is more restrictive than the bandwidth required to supply the pipeline. The resource utilisation of the FPGA could be reduced by using BRAMs in addition to distributed memory. The probability of movement memory block inside each movement voxel is taking slightly more than half the resources of the FPGA fabric. With some modifications to the movement voxel and MPE structure, it should be possible to reduce the number of probability of movement memory blocks to a single one per MPE, instead of one in every voxel, similarly to how the reaction selector is used in the reaction phase. The movement update would have to be slightly more complex, but the difference in size is probably worth it. A stronger optimisation of the voxels could also lead to a more efficient implementation.

The bandwidth utilisation, even if not a major bottleneck, could be improved by reducing the amount of data spent on transferring buffered voxel. For example, a brute force approach where the full buffered neighbourhood is received and sent along with the processed data uses 26% of the bandwidth for the buffered data when using a 4x4 array of PEs. As the number of PEs increase, the layer of buffering voxels becomes less important. For example, it reduces to 15% with a 8x8 array of PEs. By storing the voxel information that will be needed again shortly on the device, a smart memory controller could considerably reduce this wasted bandwidth at the cost of increased memory capacity and additional logic. Both the bandwidth and the resource utilisation scale linearly with the size of the system.

Besides increasing the number of PEs on the board, the throughput could also be improved by increasing the clock frequency of the pipeline. The best way to do so is to further pipeline the critical data path until the increase of clock frequency does not compensate for the extra number of stages needed to update the stage. Currently, adding one extra clock cycle per voxel increases the number of steps by roughly 10%. Thus, as long as the increase in clock speed is larger than 10% the throughput will increase.

CHAPTER 5
Future Research Work

The development of both the algorithm and the architecture is the first attempt at building a hardware-accelerated, scalable, stochastic simulator which includes locality and crowding effects. While this specific goal has been met, several aspects of the simulator could be improved or modified such as adding more functionality or making it easier to use, faster, and biologically more accurate. In this section, we will look at the possible avenues that can be pursued to improve GridCell.

5.1 Complete VHDL Implementation

The hardware implementation into an actual FPGA platform is the main short-term goal of the research work. The algorithm of GridCell was developed with the idea that it would be parallelised. An efficient pipelined and parallel architecture was designed and synthesised. The next step is to physically implement the pipeline into an actual accelerating hardware platform. A streaming approach would be used where the three different stages consisting of data loading, unloading and algorithm execution are overlapped. In order to do this, a dual buffering strategy is adopted. Two different SRAM banks connected to the FPGA are required, the first one storing the input data and the second one containing the output data. Each bank is divided into two sections. While one section from the input RAM feeds the data to the algorithm, the other section is receiving the data to be used in the next iteration from the host. Meanwhile, one section from the output RAM is receiving the processed data from the algorithm block while the other section is sending the processed data from the previous iteration to the host. When both sections in each RAM banks have finished doing their respective task, the sections are swapped and the next iteration can start. Figure 5–1 demonstrates a schematic of a classical dual buffering streaming model.

Compared to a constructive approach where the full simulation space is represented inside the FPGA, the streaming approach removes the dependency issues between the size of the FPGA and the size of the simulation. A larger simulation simply takes more iterations to complete, and modifying the size of the simulation does not require any modification to the pipeline.

The SGI RC-100 is a good example of a platform with high bandwidth between the host and the FPGA and it already supports streaming applications inherently. The SRAM blocks connecting to the FPGA are dual-ported and can perform the ping-pong buffering. The tools and FPGA

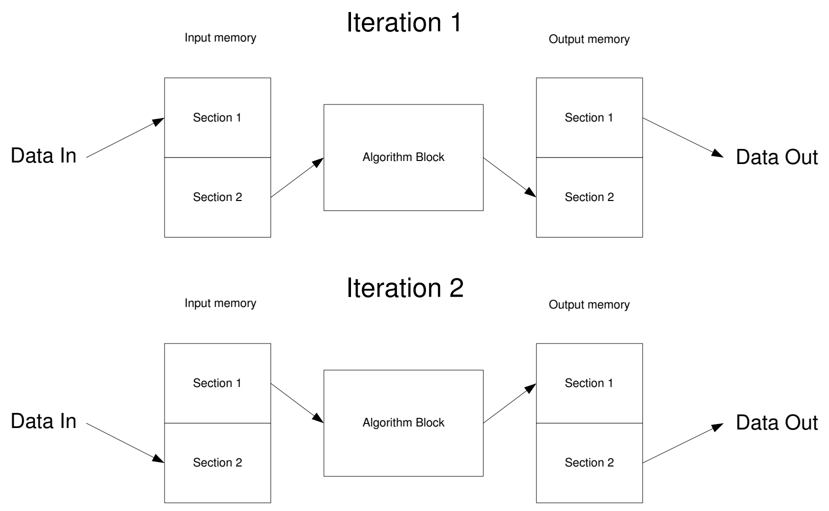

Figure 5–1: Schematic view of the dual buffering technique.

core elements are supplied by the provider to interface between the host system and the FPGA algorithm. Therefore, the memory controller dealing with the timings of the memory banks and the bus interfacing between the FPGA and the host CPU does not have to be manually developed. The hardware platform has a decent bandwidth of 6.4 GB/s between the host and the FPGA, which is large enough to support speedups of more than two orders of magnitude. The system can also contain several FPGA devices.

Finally, an input/output stage interfacing between the SRAM port and the pipeline data format is required. Theses two stages serve a similar purpose as serial to parallel converters and parallel to serial converters. The input buffering stage reads the SRAM port (a 128-bit bus in the RC-100 architecture) divides the bus data into voxel data and aligns each voxel so that they can be read by the first stage of the pipeline. The output buffering stage performs the reverse operation. Assuming 16-bit voxels, a 128-bit bus would contain exactly 8 voxels. Using voxel sizes that are not multiples of 128 would require a slightly more complex alignment.

5.2 Scaling to Multiple FPGAs

The architecture is scalable to multiple FPGAs without any modification to the pipeline. Adding more FPGAs increases the number of processing units in the pipeline, so that each iteration of the pipeline covers more area. Since the computational load is evenly divided between the FPGAs, the speed increases linearly with the number of FPGAs used. When multiple FPGAs are used, the host CPU processes the central buffer zone between the FPGAs sections while each FPGA takes care of its own section. The central buffer zone plays a similar role to the buffer zone surrounding each individual pipeline. The FPGA can move particles into the buffer zone or use it to find reactants for a reaction, but they do not actually process the voxels inside the zone, which is why the CPU has to perform the update process. By processing the voxels adjacent to the central buffer zone first, the FPGA and the CPU can work concurrently. Indeed, once the FPGAs have produced their results and written them back into memory, the software can start processing the nearby buffer zone. An example of the update process with four FPGAs is shown in Figure 5–2. This update process can be modified for any number of devices. The architecture is also scalable to multiple CPUs in a similar fashion, where each CPU can be affected to a given zone. One could also imagine a system with a master/slave hierarchy where the master CPU takes care of the data transfer and communication, while the slave CPUs work on their given zone. The central buffer zone is fairly small compared to the full simulation volume, and the proportion of voxels to be processed by the CPU gets smaller and smaller as the simulation space increase. For example, in a system using four FPGAs and a single CPU containing 10^7 and 10^9 voxels, the central buffer zone amounts to respectively 1.5% and 0.4% of the whole simulation space. The main advantage of using this hybrid CPU/FPGA solution is the complete removal of any direct communication between the FPGAs, all the communication being handled by the central CPU.

5.3 Possible Improvements to GridCell

This section discusses some of the possible modifications and improvements which would not request significant modifications to the update algorithm and, consequently, to the VHDL architecture. These improvements can be classified into several categories: user interface modifications, input/output improvements, stronger SBML support, high definition compartments and particles and grid modifications.

The current user interface could use several improvements to make it more user friendly and many functionalities could be added or improved. For example, adding a model editor where the user could build the biological models graphically would remove the need to manually edit and tinker the SBML file. Adding the options to do a steady state or sensitivity analysis would also be useful. Adding a command line interface would allow for an easier use of automatic script events. A check point feature would also enable to save and load simulations in the middle of their execution. The ability to automatically link to Internet biological databases could also be implemented. Improving the 3D graphics could be done and adding a "hot spot" feature which would locate the high density areas of reactions sounds like a nice idea. The list can go on.

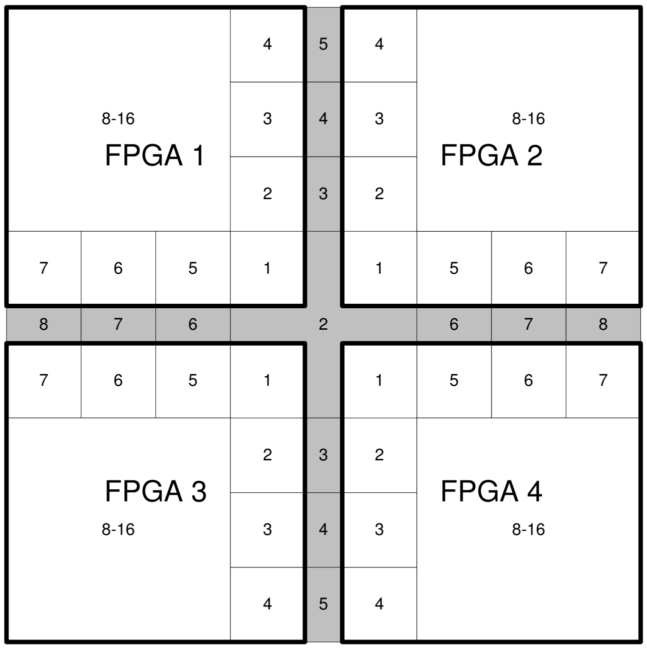

Figure 5–2: Zone partitioning with multiple FPGAs working in parallel. In this example, we assume four FPGAs and one CPU. The simulation space is divided into four equal parts for each FPGA and a central buffer zone updated by the CPU (shaded area). Each FPGA starts by processing the portion of their zone closest to the centre of the simulation space (iteration 1). During iteration 2, the FPGAs update a nearby zone adjacent to the buffer zone, denoted by 2, while the CPU can start updating the voxels in the buffer zone 2. After the 7th iteration, all voxels in the buffer zone can be updated.

While it is possible to construct relatively interesting compartment shapes from the superposition of multiple rectangular shapes, the shape feature is fairly limited in itself and could be improved. A big improvement would be to design the compartments using the constructive solid geometry technique. CSG is a technique allowing the creation of complex shapes by performing a series of simple operations combining simpler objects. The standard operations are union, intersection, addition and subtraction, along with the possibility to rotate and translate each individual construct. The final shape would then be voxelised and included in the simulation model. It could also be possible to push the idea a bit further and implement the ability to load a discretised version of compartments from real 3D images.

In the same vein as the high definition compartments, adding the support of large, complex and non regular particles would improve the biological accuracy of GridCell. These complex particles could have different binding points with different properties and shapes and could be used to emulate localisation and structural effects. Representing DNA particles with high precision is a good example where this new technology could be useful. Unfortunately, due to limitations of the algorithm, these large molecules would not be able to move. As the largest molecules tend to move slower than the smaller particles, however, the non-moving state of the very large particle might actually not be such a bad assumption for the modelling purpose.

SBML is a modular model description language which offers a lot of flexibility. GridCell is currently supporting the basic set of features allowing the description of the modelling space. It does not however support every possible representation within the SBML specification. As such, some modifications to the file might be needed to convert a biological model coming from an external simulator, although this is not unique to GridCell, since other tools have to do this as well. It could be possible to reduce the amount of modifications by expanding the number of supported features. For example, the mathematical expression parsing and unit definition could be stronger and more flexible. Another improvement would be the addition of the support of SBML level 3 (level 2 revision 3 is currently supported), which was released earlier this year. Finally, when the specifications for geometry and spatial effects will be released, supporting these features would allow the sharing of spatial models with other simulators without tinkering with custom-made annotations.

GridCell currently uses a grid of type D3Q27. This type of grid is usually seen as fairly complex and memory expensive [56]. In the FPGA implementation, the full 27-directional grid is not physically implemented. The grid is instead emulated by using a cubic grid where the memory accesses are divided into a cascade of simpler steps spanning over multiple clock cycles. The D3Q27 grid can be considered as having a radius equal to three when accessing the farthest corners of the neighbourhood. Modifying the grid type to D3Q19 would reduce the neighbourhood radius to two, which would reduce the maximum access time from three clock cycles to two clock cycles. This would reduce the number of clock cycles of the movement update by two and the reaction update by three, a reduction of 33% in latency. Although some speed gain is expected, modifying the grid would not, however, result in a direct speed up of 1.5x since the architecture already needs extra clock cycles to pipeline the longer update stage and increase the clock speed. The modification of the grid would also impact the biological behaviour and accuracy of the simulation. However, most LBM applications that use this kind of grid are using the simpler D3Q19 grid as the loss in precision and accuracy is considered insignificant compared to the reduction in complexity [56]. As both LBM and GridCell algorithms share a lot of similarities in their structure, it would make sense that modifying the Grid to a D3Q19 would also not have a large impact on the biological accuracy of the simulations. Exploring this avenue to quantify the speed improvements and modification in biological behaviour is therefore relevant.

5.4 Limitation of Current Algorithm

From a biological accuracy point of view, the fact that all moving particles are represented as occupying the same volume is one of the biggest limitations of the algorithm. We know that in living cells, the smallest particles such as ions are several order of magnitudes smaller than the largest molecules, such as DNA. As all the particles have the same size, they forcibly also have the same shape, which is also a fairly large biological assumption. The artefacts caused by this assumption can be limited by using a voxel size which is the estimate of the average size of the proteins taking place in the simulation. Particles that are assumed to be larger should also be assigned a slower diffusion speed than smaller particles. While the GridCell model is more accurate than the dimensionless particle representation often used in today's simulations, the ability to model differently sized and shaped particles would improve the biological accuracy of the simulation.

All particles in GridCell follow a Brownian random walk. This type of movement is accurate for most particles in a biological system; it is not however, the only way molecules move in the cell. While membrane active transport can be simulated by specifying different rates of permeability when entering or exiting a compartment, other active transport effects, such as molecular motors or transport tubes, cannot be simulated accurately.

Limiting the movement to the neighbouring voxels has many advantages from a computational point of view, such as the removal of a search algorithm or the need to handle probability density functions. However, in order to respect the specified diffusion speed assigned to the species, an upper limit on the time steps size is imposed. Depending on the range of parameters used, the length of the time step could be limited by the diffusion speed which would affect the overall speed of the simulation.

Finally, representation of molecular species with a mechanical function such as the cytoskeleton and coatomers (molecules that curve and stabilise membranes) or transport tubes are a great challenge to represent in the current context.

5.5 Implementation on Different Architectures

The FPGA implementation was selected for many reasons. One of the main reasons was the amount of freedom provided with the inherent customisation of the FPGA that is not available with other accelerating platforms. It was possible to use the spatial parallelism and large internal memory bandwidth provided by the FPGAs to design a highly efficient pipelined architecture made of several processing elements. However, FPGAs are not the only type of accelerating hardware platform available and GridCell could respond well to a different architecture. A interesting alternative would be to explore even further how well it responds to other parallel accelerating platforms such as a GPU, a computer cluster or the Cell Broadband Engine (CBE)[25][1].

5.6 Next Generation GridCell

In order to overcome the above-mentioned limitations of the current revision of GridCell, a second generation GridCell can be developed. The current approach focuses on simple and regular kinetics in order to have an efficient and regular brute-force parallel architecture. By building a regular grid, it is possible to process all the particles in parallel with simple logic. However, it also puts some limitations on the behaviour, size and shape of the particles. A second generation GridCell could add a more accurate representation of particle size, such as by spanning a single particle over multiple grid units. With this feature, it would be possible to implement much more realistic reactions, like the formation of complex resulting in the attachment of the reactants instead of collapsing them together. The different parts of a molecule could also be assigned to a different function such as a binding or reaction site. Movement and collision detection and crowding effects would also be more accurately simulated with complex particles and, we could even integrate the notion of momentum in the simulator.

This algorithm would invariably be much more computationally expensive than the current version and would also need to be accelerated with parallel hardware. Also, the modifications to the algorithm to simulate large scale particles would require an extensive overhaul of the proposed architecture. The application of the CA model is very well suited for dealing with individual particles, but CA algorithms are somewhat restricted when dealing with the large range communication that would be required for movement and collision detection of large molecules. An accelerated version of the second generation GridCell would probably benefit more from a particle per particle processing architecture, similarly to n-body problems, instead of the CA model currently used, which processes every voxels at all time. Both approaches should be explored more thoroughly.

CHAPTER 6
Conclusion

We have explored the field of study of systems biology, which thrives to understand quantitatively the biological systems. The sheer amount of information and the large differences in the scales of the size, time and number of elements describing those systems make them impossible to analyse without external computerised assistance. Over the past decade and a half, with the exponential growth of computer power and data retrieving methods, several tools have been developed to gain additional insight on various types of biological networks. Between the top-end applications, where data is linked together via signalling and regulatory pathways into large-scale systems, and bottom-end applications, which work on the molecular dynamics of the particles, a family of middle-end range simulators has made its apparition. These middle-range simulators, explore the stochastic effects of particles, the heterogeneity of the space and the crowding effects, which are all affecting the behaviour of living cells. One of the goal of those simulators, and of systems biology as a whole, is to simulate a complete cell with enough details to capture the internal logic describing its behaviour.

We have presented a novel stochastic simulator based on the cellular automata model. The simulator supports several key aspects of biological behaviour inside a living cell, such as spatial effects, diffusion, compartment supports, explicit molecular representation, individual microscopic particle tracking and of course, stochasticity. Other key aspects of the simulator include SBML support and interactive real time user interface. The CA model results in an efficient algorithm which ensures a simple collision detection scheme, eliminates the need to perform expensive particle searches for reactions and avoids the use of floating points calculation during the update phase. A regular cubic grid of type D3Q27 is adopted where particles move and reacts in discrete location with discrete time step. The particles diffuse through space by following Brownian dynamics at a rate derived from the Einstein-Smoluchowski equations. The particles reaction rates are derived from the macroscopic law of mass action, which implies that under well-mixed assumption, GridCell matches results obtained by the SSA algorithm, which is an exact numerical realisation of the CME describing the state of the system. However, GridCell's main purpose is not to emulate the SSA algorithm under the well-mixed approximation but, on the contrary, to explore the behaviour of the systems under heterogeneous and crowded environments. Simulations of crowded system showed that after a certain amount of crowdiness, the enzyme particles of the Michaelis-Menten were in a state called diffusion-limited, and the overall reaction rate was affected. As the system was becoming more and more crowded, the reaction rate was becoming slower and slower. An explicit crowding representation, where particles occupy a physical location and volume, is essential in simulating crowding effects and is one of the key advantages of GridCell compared to other simulators, as most of them consider particles as dimensionless entities. Spatial effects are also of prime importance and are believed to play an important role in the stability and behaviour of signalling pathways in biological systems. A system describing a semi-porous membrane with localised enzyme sites was simulated under GridCell. The behaviour of the system with spatial effects was found to be significantly different than the well-mixed version of the same system.

The GridCell software program is available on the Web at www.iml.ece.mcgill.ca/GridCell/. It was first released on September 2008, and a revision implementing several bug fixes, UI improvement and compartment support was released on February 2010. It comes in a package with a few biological systems that are ready to be simulated, as well as a user manual detailing how to use GridCell.

The major problem with biological simulators is that the systems they are trying to describe are inherently very large, complex and often unpredictable. The amount of processing power needed to describe them is a few orders of magnitude above the current computational power by conventional means, and this is true for the serial implementation of the GridCell algorithm as well. The gap between available processing power and the complexity of the system can be reduced by parallelising the algorithm. FPGAs can provide large speedups to the application, which can benefit from spatial parallelism, pipelining and high internal memory bandwidth. GridCell highly benefits from these three properties, and a FPGA architecture exploiting them has been designed. The architecture uses a streaming approach with six pipelined stages to increase the throughput and make the solution scalable. Each stage contains multiple processing units and operates on several voxels at the same time. When compared to the serial processor implementation, gain in performance larger than 75x were achieved. The solution is also scalable to multiple FPGA devices to increase the speedup range to an even higher level.

The development of highly scalable biological simulators and their accelerated architectures are a really interesting and exciting area of research. The multidisciplinary nature of the research work involves expertise in several different branches of science such as biology for the general knowledge and principles of life, systems biology in order to use and design a valid biological simulator, software engineering to implement the simulator into an actual computer program and the conception of the user interface and, finally, computer/electrical engineering for the design of the VLSI implementation into an accelerated hardware architecture.

The development of scalable and biologically accurate simulators, as well as the construction of biological model grasping the behaviour of full living cells, is expected to assist drugs development as well improving our understanding of diseases and their possible treatments. Due to the inherently large scale and high complexity of biological systems, parallel processing is expected to take an important role in the simulation of those systems and it was shown that speedups larger than one order of magnitude can be gained with massively parallel architectures. The development of hierarchical multi-scale systems, where each level provides a different amount of spatial and temporal resolution, is also believed to be key in achieving the not so long-term goal of full cell simulation in real time.

References

[1] S. Al Assaad. Biochemical system simulation on a heterogeneous multicore processor. Master's thesis, McGill University, Montreal, 2008.

[2] M. Ander et al. Smartcell, a framework to simulate cellular processes that combines stochastic approximation with diffusion and localisation: analysis of simple networks. *Syst. Biol.*, 1:129–138, 2004.

[3] S. S. Andrews and D. Bray. Stochastic simulation of chemical reactions with spatial resolution and single molecule detail. *Phys. Biol.*, 1:137–151, 2004.

[4] N. Azizi, I. Kuon, A. Egier, A. Darabiha, and P. Chow. Reconfigurable molecular dynamics simulator. *Symposium on Field-Programmable Custom Computing Machines (FCCM'04)*, pages 197–204, April 2004.

[5] E. Berlekamp, J. H. Conway, and R. Guy. *Winning Ways for Your Mathematical Plays*, volume 2. Academic Press, 1982.

[6] L. Boulianne, S. Al Assaad, M. Dumontier, and W. J. Gross. Gridcell: A stochastic particle-based biological system simulator. *BMC Systems Biology*, 2(66), July 2008.

[7] L. Boulianne, M. Dumontier, and W. J. Gross. A stochastic particle-based biological system simulator. In *Proceedings of the Summer Computer Simulation Conference*, pages 794–801. San Diego, California (USA), 2007.

[8] G. Broderick, M. Ru'aini, E. Chan, and Ellison M. J. A life-like virtual cell membrane using discrete automata. *In Silico Biology*, 5(2):163–178, 2005.

[9] A. W. Burk. Essays on cellular automata. Technical report, University of Illinois, Urbana, 1970.

[10] G. Cappuccino and G. Cocorullo. Custom reconfigurable computing machine for high performance cellular automata processing. *TechOnLine Publication*, July 2001.

[11] M. Chiu, M. C. Herbordt, and M. Langhammer. Performance potential of molecular dynamics simulations on high performance reconfigurable computing systems. *Proceedings of International Workshop on High-Performance Reconfigurable Computing Technology and Applications (HPRCTA 2008)*, 2008.

[12] J. G. Duman, L. Chen, and Hille B. Calcium transport mechanisms of pc12 cells. *Gen. Physiol.*, 131:307–323, 2008.

[13] C. P. Fall and J. E. Keizer. *Computational Cell Biology*, chapter Dynamic Phenomena in Cells. Springer-Verlag, New York;, 2002.

[14] K. M. Franks and T. J. Sejnowski. Complexity of calcium signaling in synaptic spines. *BioEssays*, 24:1130–1144, 2002.

[15] U. Frisch, B. Hasslacher, and Y. Pomeau. Lattice-gas-automata for the navier-strokes equation. *Physical Review Letters*, 56(14):1505–1508, April 1986.

[16] A. B. Fulton. How crowded is the cytoplasm? *Cell*, 30(2):345–347, 1982.

[17] A. Funahashi, Y. Matsuoka, A. Jouraku, M. Morohashi, N. Kikuchi, and H. Kitano. Celldesigner 3.5: A versatile modeling tool for biochemical networks. In *Proceedings of the IEEE Volume 96, Issue 8*, pages 1254–1265, 2008.

[18] D. T. Gillespie. A general method for numerically simulating the stochastic time evolution of couple chemical reactions. *Journal of Computational Physics*, 22:403–434, 1976.

[19] J. S. Goggan et al. Evidence for ectopic neurotransmission at a neuronal synapse. *Science*, 15:446–451, 2005.

[20] R. Grima and S. Schnell. A mesoscopic simulation approach for modeling intracellular reactions. *Journal of Statistical Physics*, in press (DOI:10.1007/s10955-006-9202-z) 2007.

[21] J. Hattne, D. Fange, and J. Elf. Stochastic reaction-diffusion simulation with mesord. *Bioinfomatics*, 21:2923–2924, 2005.

[22] J. L. Hennessy and D. A. Patterson. *Computer Architecture A Quantitative Approach.* Morgan Kaufmann Publishers, 3rd edition edition, 2003.

[23] C. Y. Huang and J. E. Ferrel. Ultrasensitivity in the mitogen-activated protein kinase cascade. *Proceedings of the National Academy of Sciences*, 93:10078–10083, September 1996.

[24] M. Hucka et al. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics*, (19):524–531, 2003.

[25] IBM. *Cell BE Programming Handbook.* 1.11 edition, May 2008. 884p.

[26] J. F. Kean, C. Bradley, and C. Ebeling. A compiled accelerator for biological cell signaling simulations. In *Proceedings of the FPGA04*, pages 233–241. Monterey, California, USA, 2004.

[27] M. Kerker. Brownian movements and molecular reality prior to 1900. *Journal of Chemical Education*, (51):764–768, 1974.

[28] B. N. Kholodenko. Four-dimensional organization of protein kinase signaling cascades: the roles of diffusion, endocytosis and molecular motors. *Exp. Biol.*, 206:2073–2082, 2003.

[29] T. Kobori, T. Maruyama, and T. Hoshino. A cellular automata system with fpga. *The 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 120–129, 2001.

[30] N. Le Novere and T. S. Shimizu. Stochsim: modelling of stochastic biomolecular processes. *Bioinformatics*, (17):575–576, 2001.

[31] C. Lemerle, B. D. Ventura, and L. Serrano. Space as the final frontier in stochastic simulations of biological systems. *FEBS Letters*, 578:1789–1794, 2005.

[32] H. Li and L. R. Petzold. Stochastic simulation of biochemical systems on the graphics processing unit. *Technical report, Department of Computer Science, University of California, Santa Barbara, 2007. Submitted.*

[33] H. Li and L. R. Petzold. Logarithmic direct method for discrete stochastic simulation of chemically reacting systems. Technical report, Department of Computer Science, University of California, Santa Barbara, 2007.

[34] K. Lipkow, S. S. Andrews, and D. Bray. Simulated diffusion of phosphorylated chey through the cytoplasm of escherichia coli. *J. Bact.*, 187:45–53, 2005.

[35] A. P. Minton. The effect of volume occupancy upon the thermodynamic activity of proteins: some biochemical consequences. *Mol Cell Biochem*, (55):119–140, 1983.

[36] M. Mitchell. Computation in cellular automata: A selected review. In T. Gramss, S. Bornholdt, M. Gross, M. Mitchell, and T. Pellizzari, editors, *Nonstandard Computation*, pages 95–140. Weinheim, Germany, 1998.

[37] S. Plimpton and A. Slepoy. Chemcell: A particle-based model of protein chemistry and diffusion in microbial cells. *Sandia Technical Report SAND2003-4509*, December 2003.

[38] C. V. Rao, D. M. Wolf, and A. P. Arkin. Control, exploitation and tolerance of intracellular noise. *Nature*, 420(6912):231–237, 2002.

[39] D. Ridgway, G. Broderick, Lopez-Campistrous A., M. Ru'aini, P. Winter, M. Hamilton, P. Boulanger, A. Kovalenko, and Ellison M. J. Coarse-grained molecular simulation of diffusion and reaction kinetics in a crowded virtual cytoplasm. *Biophysical Journal*, 94:3748–3759, May 2008.

[40] D. Ridgway, G. Broderick, and M. J. Ellison. Accommodating space, time and randomness in network simulation. *Current Opinion in Biotechnology*, 17:493–498, 2006.

[41] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, and S. Vo. *A Statistical Test Suite for the Validation of Random Number Generators and Pseudo Random Number Generators for Cryptographic Applications.* NIST, http://csrc.nist.gov/groups/ST/toolkit/rng/index.html, April 2010. Special Publication 800-22rev1a.

[42] L. Salwinski and D. Eisenberg. In silico simulation of biological network dynamics. *Nature Biotechnology, Volume 22, Number 8*, pages 1017–1019, 2004.

[43] C. Sanford, M. L. K. Yip, C. White, and J. Parkinson. Cell++simulating biochemical pathways. *Bioinfomatics*, 22:2918–2925, 2006.

[44] J. Schaff et al. A general computational framework for modeling cellular structure and function. *Journal of Biophys.*, (73):1135–1146, November 1997.

[45] S. Schnell and T. E. Turner. Reaction kinetics in intracellular environments with macromolecular crowding: simulations and rate laws. *Progress in Biophysics and Molecular Biology*, 85:235–260, 2004.

[46] Canadian Industry Statistics Development Team Web Site. Pharmaceutical and medicine industry (sic - 3741). January 1998.

[47] L. L. Smith. Transims home page. 2002.

[48] J. Soto. Statistical testing of random number generators. *Proceedings of the 22nd National Information Systems Security Conference*, 1999.

[49] J. R. Stiles and T. M. Bartol. Monte carlo methods for simulating realistic synaptic microphysiology using mcell. In E. DeSchutter, editor, *Computational Neuroscience: Realistic Modeling for Experimentalists*, pages 87–127. CRC Press, Boca Raton, 2001.

[50] Z. Szallasi, J. Stelling, and V. Periwal. *System Modeling in Cellular Biology : From concepts to nuts and bolts.* ISBN 0-262-19548-8. MIT Press, 2006.

[51] K. Takahashi, S. N. V. Arjunan, and M. Tomita. Space in systems biology of signaling pathways - towards intracellular molecular crowding in silico. *FEBS Letters*, 579:1783–1788, 2005.

[52] I. Tremmel, H. Kirchhoff, E. Weis, and G. Farquhar. Dependence on plastoquinol diffusion on the shape, size and density of integral thylakoid proteins. *Biochim Biophys Acta*, 1607(2-3):97–109, 2003.

[53] J. L. Tripp, H. S. Mortveit, A. A. Hansson, and M. Gokhale. Metropolitain road traffic simulation on fpgas. *13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'05)*, pages 117–126, 2005.

[54] T. E. Turner, S. Schnell, and K. Burrage. Stochastic approaches for modelling in vivo reactions. *Computational Biology and Chemistry*, 28:165–178, 2004.

[55] T. VanCourt and M. C. Herbordt. Application-dependent memory interleaving enables high performance in fpga-based grid computations. *IEEE Symposium on Field-Programmable Custom Computing Machines*, 2006.

[56] X. Wei, L. Wei, K. Mueller, and A. E. Kaufman. The lattice-boltzmann method for simulating gaseous phenomena. *IEEE Transactions on Visualization and Computer Graphics*, 10(2):164–176, March/April 2004.

[57] J. R. Weimar. Cellular automata approaches to enzymatic reaction networks. In S. Bandini, B. Chopard, and M. Tomassini, editors, *5th International Conference on Cellular Automata for Reasearch and Industry ACRI*, pages 294–303, Switzerland, 2002. Springer.

[58] E. W. Weisstein. Elementary cellular automaton. From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/ElementaryCellularAutomaton.html.

[59] S. Williams, J. Carter, L. Oliker, J. Shalf, and K. Yelick. Lattice boltzmann simulation optimization on leading multicore platforms. *International Parallel and Distributed Processing Symposium (IPDPS)*, 2008.

[60] L. Yang and P. A. Iglesias. Modeling spatial and temporal dynamics of chemotactic networks. *Chemotaxis: Methods and Protocols, Methods in Molecular Biology*, 574:489–505, 2009.

[61] M. Yoshimi, Y. Osana, T. Fukushima, and H. Amano. Stochastic simulation for biochemical reactions on fpga. In *in Proc. of FPL'04*, pages 105–114, 2004.

[62] Y. Zhao. Lattice boltzmann based pde solver on the gpu. *The Visual Computer*, 24(5):323–333, May 2008.

[63] S. B. Zimmerman and A. P. Minton. Macromolecular crowding: biochemical, biophysical, and physiological consequences. *Annu. Rev. Biophys. Biomol. Struct.*, 22:27–65, 93.